551,891

(12) NACH DEM VERTRAG ÜBER DIE INTERNATIONALE ZUSAMMENARBEIT AUF DEM GEBIET DES PATENTWESENS (PCT) VERÖFFENTLICHTE INTERNATIONALE ANMELDUNG

(19) Weltorganisation für geistiges Eigentum
Internationales Büro

(43) Internationales Veröffentlichungsdatum
14. Oktober 2004 (14.10.2004)

PCT

(10) Internationale Veröffentlichungsnummer
WO 2004/088502 A2

(51) Internationale Patentklassifikation[7]: G06F 9/00

(21) Internationales Aktenzeichen: PCT/EP2004/003603

(22) Internationales Anmeldedatum:
5. April 2004 (05.04.2004)

(25) Einreichungssprache: Deutsch

(26) Veröffentlichungssprache: Deutsch

(30) Angaben zur Priorität:
103 15 295.4 4. April 2003 (04.04.2003) DE
103 21 834.3 15. Mai 2003 (15.05.2003) DE

(71) Anmelder *(für alle Bestimmungsstaaten mit Ausnahme von US)*: **PACT XPP TECHNOLOGIES AG** [DE/DE]; Muthmannstrasse 1, 80939 München (DE).

(72) Erfinder; und
(75) Erfinder/Anmelder *(nur für US)*: **VORBACH, Martin** [DE/DE]; Gotthardstrasse 117a, 80689 München (DE).

(74) Anwalt: **PIETRUK, Claus, Peter**; Heinrich-Lilienfein-Weg 5, 76229 Karlsruhe (DE).

(81) Bestimmungsstaaten *(soweit nicht anders angegeben, für jede verfügbare nationale Schutzrechtsart)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Bestimmungsstaaten *(soweit nicht anders angegeben, für jede verfügbare regionale Schutzrechtsart)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), eurasisches (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), europäisches (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Veröffentlicht:
— *ohne internationalen Recherchenbericht und erneut zu veröffentlichen nach Erhalt des Berichts*

Zur Erklärung der Zweibuchstaben-Codes und der anderen Abkürzungen wird auf die Erklärungen ("Guidance Notes on Codes and Abbreviations") am Anfang jeder regulären Ausgabe der PCT-Gazette verwiesen.

(54) Title: METHOD AND DEVICE FOR DATA PROCESSING

(54) Bezeichnung: VERFAHREN UND VORRICHTUNG FÜR DIE DATENVERARBEITUNG

(57) Abstract: The invention relates to a data processing device with a data processing logic cell field and at least one sequential CPU, wherein a coupling of the sequential CPU to the data processing logic cell field, for data exchange, particularly in block form, by means of lines leading to a cache memory is provided.

(57) Zusammenfassung: Die Erfindung betrifft eine Datenverarbeitungsvorrichtung mit einem Datenverarbeitungslogikzellenfeld und zumindest einer Sequenziell-CPU. Hierbei ist vorgesehen, dass eine Ankopplung der Sequenzielll-CPU und des Datenverarbeitungslogikzellenfeldes zum Datenaustausch in insbesondere blockweiser Form durch zu einem Cache-Speicher führende Leitungen möglich ist.

WO 2004/088502 A2

Titel: Verfahren und Vorrichtung für die Datenverarbeitung

Beschreibung

Die vorliegende Erfindung betrifft das oberbegrifflich Beanspruchte und befasst sich somit mit Verbesserungen bei der Verwendung von rekonfigurierbaren Prozessortechnologien für die Datenverarbeitung.

Verwiesen wird bezüglich des bevorzugten Aufbaus von Logikzellenfeldern auf die XPP-Architektur und vorveröffentlichte sowie jüngere Schutzrechtsanmeldungen des vorliegenden Anmelders, die zu Offenbarungszwecken vollumfänglich eingegliedert sind. Erwähnt seien somit insbesondere die DE 44 16 881 A1, DE 197 81 412 A1, DE 197 81 483 A1, DE 196 54 846 A1, DE 196 54 593 A1, DE 197 04 044.6 A1, DE 198 80 129 A1, DE 198 61 088 A1, DE 199 80 312 A1, PCT/DE 00/01869, DE 100 36 627 A1, DE 100 28 397 A1, DE 101 10 530 A1, DE 101 11 014 A1, PCT/EP 00/10516, EP 01 102 674 A1, DE 198 80 128 A1, DE 101 39 170 A1, DE 198 09 640 A1, DE 199 26 538.0 A1, DE 100 50 442 A1, sowie die PCT/EP 02/02398, DE 102 40 000, DE 102 02 044, DE 102 02 175, DE 101 29 237, DE 101 42 904, DE 101 35 210, EP 01 129 923, PCT/EP 02/10084, DE 102 12 622, DE 102 36 271, DE 102 12 621, EP 02 009 868, DE 102 36 272, DE 102 41 812, DE 102 36 269, DE 102 43 322, EP 02 022 692, ebenso wie die EP 02 001 331 und die EP 02 027 277.

Ein Problem bei herkömmlichen Ansätzen zu rekonfigurierbaren Technologien besteht dann, wenn die Datenverarbeitung primär

auf einer sequenziellen CPU unter Hinzuziehung eines konfigurierbaren Datenverarbeitungslogikzellenfeldes oder dergleichen erfolgen soll und/oder eine Datenverarbeitung gewünscht ist, in der viele und/oder umfangreiche sequenziell auszuführende Verarbeitungsschritte vorliegen.

Es sind Ansätze bekannt, die sich damit befassen, wie eine Datenverarbeitung sowohl auf einem konfigurierbaren Datenverarbeitungslogikzellenfeld als auch auf einer CPU erfolgen kann.

So ist aus der WO 00/49496 ein Verfahren zum Ausführen eines Computerprogrammes mit einem Prozessor bekannt, der eine konfigurierbare funktionelle Einheit umfasst, die in der Lage ist, rekonfigurierbare Anweisungen auszuführen, deren Effekt zur Laufzeit durch Laden eines Konfigurationsprogrammes redefiniert werden kann, wobei das Verfahren die Schritte umfasst, daß Kombinationen rekonfigurierbarer Anweisungen ausgewählt, ein respektives Konfigurationsprogramm für jede Kombination erzeugt und das Computerprogramm ausgeführt wird. Dabei soll jedes Mal, wenn eine Anweisung aus einer der Kombinationen während der Ausführung gebraucht wird und die konfigurierbare funktionelle Einheit nicht mit dem Konfigurationsprogramm für diese Kombination konfiguriert ist, das Konfigurationsprogramm für alle der Anweisungen der Kombination in die konfigurierbare funktionelle Einheit geladen werden. Weiter ist aus der WO 02/50665 A1 eine Datenverarbeitungsvorrichtung mit einer konfigurierbaren funktionellen Einheit bekannt, wobei die konfigurierbare funktionelle Einheit dazu dient, eine Anweisung gemäß einer konfigurierbaren Funktion auszuführen. Die konfigurierbare funktionelle Einheit weist eine Vielzahl von unabhängigen konfigurierbaren Logikblöcken

zum Ausführen programmierbarer Logikoperationen auf, um die konfigurierbare Funktion zu implementieren. Konfigurierbare Verbindungsschaltkreise sind zwischen den konfigurierbaren Logikblöcken und sowohl den Eingängen als auch den Ausgängen der konfigurierbaren funktionellen Einheit vorgesehen. Dies erlaubt eine Optimalisierung der Verteilung von Logikfunktionen über die konfigurierbaren Logikblöcke.

Ein Problem bei herkömmlichen Architekturen besteht dann, wenn eine Ankopplung erfolgen soll und/oder Technologien wie Datastreaming, Hyperthreading, Multithreading und so weiter in sinnvoller und Performance steigernder Weise ausgenützt werden sollen. Eine Beschreibung einer Architektur findet sich in „Exploiting Choice: Instruction Fetch and Issue on Implementable Simultaneous Multi-Threading Processor", Dean N. Tulson, Susan J. Eggers et al, Proceedings of the 23th annual international Symposium on Computer Architecture, Philadelphia, May 1996.

Die Hyperthreading- und Multithreading-Technologien sind im Hinblick darauf entwickelt worden, dass moderne Mikroprozessoren ihre Leistungsfähigkeit aus vielen -spezialisierten und tiefpipelineartig angesteuerten funktionellen Einheiten und hohen Speicherhierarchien gewinnen, was hohe Frequenzen in den Funktionskernen erlaubt. Durch die streng hierarchischen Speicheranordnungen gibt es jedoch bei Fehlzugriffen auf Caches auf Grund des Unterschieds zwischen Kern- und Speicherfrequenzen größere Nachteile, da viele Kerntaktzyklen vergehen, bis Daten aus dem Speicher ausgelesen sind. Zudem treten Probleme auf bei Verzweigungen und insbesondere falsch vorhergesagten Verzweigungen. Es ist daher vorgeschlagen worden, als sogenanntes SMT, simultaneous multi-threading-

Verfahren zwischen verschiedenen Tasks immer dann zu wechseln, wenn eine Anweisung nicht ausgeführt werden kann oder nicht alle funktionellen Einheiten verwendet.

Die beispielhaft erwähnte Technologie der vorzitierten Nicht-Anmelder-Dokumente zeigt etwa eine Anordnung, bei der zwar Konfigurationen in ein konfigurierbares Datenverarbeitungslogikzellenfeld geladen werden können, bei welchen allerdings der Datenaustausch zwischen der ALU der CPU und dem konfigurierbaren Datenverarbeitungslogikzellenfeld, sei es ein FPGA, DSP oder dergleichen, über die Register erfolgt. Mit anderen Worten müssen Daten aus einem Datenstrom zunächst sequenziell in Register geschrieben werden und dann sequenziell wieder in diesen abgelegt werden. Auch ist ein Problem dann gegeben, wenn ein Zugriff auf Daten von extern erfolgen soll, da selbst dann noch Probleme beim zeitlichen Ablauf der Datenverarbeitung im Vergleich zur ALU und bei der Zuweisung von Konfigurationen und so weiter bestehen. Die herkömmlichen Anordnungen, wie sie aus den Nicht-Anmelder-eigenen Schutzrechten bekannt sind, werden unter anderem dazu verwendet, Funktionen im konfigurierbaren Datenverarbeitungslogikzellenfeld, DFP, FPGA oder dergleichen abzuarbeiten, die nicht effizient auf der CPU-eigenen ALU abzuarbeiten sind. Damit wird das konfigurierbare Datenverarbeitungslogikzellenfeld praktisch verwendet, um benutzerdefinierte Opcodes zu ermöglichen, die eine effizientere Abarbeitung von Algorithmen ermöglichen, als dies auf dem ALU-Rechenwerk der CPU ohne konfigurierbare Datenverarbeitungslogikzellenfelderunterstützung möglich wäre.

Im Stand der Technik ist, wie erkannt wurde, die Ankopplung demnach im Regelfall wortbasiert, nicht jedoch blockbasiert, wie es zur datenströmenden Verarbeitung erforderlich wäre. Es

ist zunächst wünschenswert, eine effizientere Datenverarbeitung zu ermöglichen, als dies mit einer engen Ankopplung über Register der Fall ist.

Eine weitere Möglichkeit zur Verwendung von Logikzellenfeldern aus grob- und/oder feingranular gebauten Logikzellen und Logikzellenelementen besteht in einer sehr losen Ankopplung eines solchen Feldes an eine herkömmliche CPU und/oder einen CPU-Kern bei eingebetteten Systemen. Hierbei kann ein herkömmliches, sequenzielles Programm auf einer CPU oder dergleichen laufen, beispielsweise ein in C, C++ oder dergleichen geschriebenes Programm, wobei von diesem Aufrufe einer Datenstromverarbeitung auf dem fein- und/oder grobgranularen Datenverarbeitungslogikzellenfeld instantiiert werden. Problematisch ist dann, dass beim Programmieren für dieses Logikzellenfeld ein nicht in C oder einer anderen sequenziellen Hochsprache geschriebenes Programm für die Datenstromabarbeitung vorgesehen werden muss. Erwünscht wäre hier, dass sowohl auf der herkömmlichen CPU-Architektur als auch auf einem mit diesen gemeinsam betriebenen Datenverarbeitungslogikzellenfeld C-Programme oder dergleichen abzuarbeiten sind, das heißt, dass insbesondere mit dem Datenverarbeitungslogikzellenfeld in quasi sequenzieller Programmabarbeitung dennoch eine Datenstromfähigkeit erhalten bleibt, während es simultan insbesondere auch möglich bleibt, dass ein CPU-Betrieb in nicht zu loser Ankopplung möglich ist. Es ist auch bereits bekannt, innerhalb einer Datenverarbeitungslogikzellenfeldanordnung, wie sie insbesondere aus PACT02 (DE 196 51 075.9-53, WO 98/26356), PACT04 (DE 196 54 846.2-53, WO 98/29952), PACT08, (DE 197 04 728.9, WO 98/35299) PACT13 (DE 199 26 538.0, WO 00/77652) PACT31 (DE 102 12 621.6-53, PCT/EP 02/10572) wobei bekannt ist, auch eine sequenzielle Datenver-

arbeitung innerhalb des Datenverarbeitungslogikzellenfeldes vorzusehen. Hierbei wird dann allerdings innerhalb einer einzelnen Konfiguration, beispielsweise um Ressourcen zu sparen, eine Zeitoptimierung zu erzielen und so weiter, eine partielle Abarbeitung erzielt, ohne dass diese bereits dazu führt, dass ein Programmierer ein Stück Hochsprachencode automatisch leicht ohne weiteres auf ein Datenverarbeitungslogikzellenfeld umsetzen kann, wie dies bei herkömmlichen Maschinenmodellen für sequenzielle Prozessoren der Fall ist. Die Umsetzung von Hochsprachencode auf Datenverarbeitungslogikzellenfelder nach Prinzipien der Modelle für sequenziell arbeitende Maschinen ist weiterhin schwierig.

Aus dem Stand der Technik ist weiter bekannt, dass mehrere Konfigurationen, die eine jeweils unterschiedliche Funktionsweise von Arrayteilen bewirken, simultan auf dem Prozessorfeld (PA) abgearbeitet werden können und dass ein Wechsel von einer oder einigen der Konfiguration(en) ohne Störung anderer zur Laufzeit erfolgen kann. Es sind Verfahren und in Hardware implementierte Mittel zu deren Umsetzung bekannt, wie sichergestellt werden kann, dass dabei ein Abarbeiten von auf das Feld zu ladenden Teilkonfigurationen ohne Deadlock erfolgen kann. Verwiesen wird hierzu insbesondere auf die die Filmo-Technik betreffenden Anmeldungen PACT05 (DE 196 54 593.5-53, WO 98/31102) PACT10 (DE 198 07 872.2, WO 99/44147, WO 99/44120) PACT13 (DE 199 26 538.0, WO 00/77652), PACT17 (DE 100 28 397.7, WO 02/13000); PACT31 (DE 102 12 621.6, WO 03/036507). Diese Technologie ermöglicht in gewisser Weise bereits eine Parallelisierung und, bei entsprechender Gestaltung und Zuordnung der Konfigurationen, auch eine Art Multitasking/Multithreading und zwar dergestalt, dass eine Planung, das heißt ein Scheduling und/oder eine Zeitnutzungs-

planungssteuerung vorgesehen ist. Es sind also aus dem Stand der Technik schon Zeitnutzungsplanungssteuerungsmittel und -verfahren per se bekannt, die, zumindest unter entsprechender Zuordnung von Konfigurationen zu einzelnen Aufgaben und/oder Fäden zu Konfigurationen und/oder Konfigurationsfolgen, ein Multitasking und/oder Multithreading erlauben. Die Verwendung solcher Zeitnutzungsplanungssteuermittel, die im Stand der Technik zur Konfigurierung und/oder Konfigurationsverwaltung verwendet wurden, zu Zwecken des Scheduling von Tasks, Threads, Multi- und Hyperthreads wird per se als erfinderisch angesehen.

Wünschenswert ist auch zumindest gemäß einem Teilaspekt in bevorzugten Varianten, moderne Technologien der Datenverarbeitung und Programmabarbeitung wie Multitasking, Multithreading, Hyperthreading unterstützen zu können, zumindest in bevorzugten Varianten einer Halbleiterarchitektur.

Der Grundgedanke der Erfindung besteht darin, Neues für die gewerbliche Anwendung bereitzustellen.

Die Lösung dieser Aufgabe wird in unabhängiger Form beansprucht. Bevorzugte Ausführungsformen finden sich in den Unteransprüchen.

Ein erster wesentlicher Aspekt der vorliegenden Erfindung ist somit darin zu sehen, dass dem Datenverarbeitungslogikzellenfeld Daten im Ansprechen auf die Ausführung einer Ladekonfiguration durch das Datenverarbeitungslogikzellenfeld zugeführt werden und/oder Daten aus diesem Datenverarbeitungslogikzellenfeld weggeschrieben (STORE) werden, indem eine STORE-Konfiguration entsprechend abgearbeitet wird. Diese La-

de- und oder Speicherkonfigurationen sind dabei bevorzugt derart auszugestalten, dass innerhalb des Datenverarbeitungslogikzellenfeldes direkt oder indirekt Adressen jener Speicherstellen generiert werden, auf welche ladend und/oder speichernd direkt oder indirekt zugegriffen werden soll. Es ist durch diese Einkonfiguration von Adressgeneratoren innerhalb einer Konfiguration möglich, eine Vielzahl von Daten in das Datenverarbeitungslogikzellenfeld einzuladen, wo sie gegebenenfalls in internen Speichern (iRAM) ablegbar sind und/oder wo sie in internen Zellen wie EALUs mit Registern und/oder dergleichen eigenen Speichermitteln abgelegt werden können. Die Lade- beziehungsweise Speicherkonfiguration ermöglicht somit ein blockweises und nahezu datenstromartiges, insbesondere gegenüber Einzelzugriff vergleichsweises schnelles Laden von Daten und es kann eine solche Lade-Konfiguration ausgeführt werden vor einer oder mehreren tatsächlich Daten auswertend und/oder verändernd abarbeitenden Konfiguration(en), mit welcher/n die vorab geladenen Daten verarbeitet werden. Das Datenladen und/oder -schreiben kann dabei typisch bei großen Logikzellenfeldern in kleinen Teilbereichen derselben geschehen, während andere Teilbereiche mit anderen Aufgaben befaßt sind. Bezüglich dieser und anderer Besonderheiten der Erfindung wird auf Fig. 1 verwiesen. Bei der in anderen veröffentlichten Dokumenten des Anmelders beschriebenen Ping-Pong-artigen Datenverarbeitung, bei der auf beiden Seiten eines Datenverarbeitungsfeldes Speicherzellen vorgesehen sind, wobei die Daten in einem ersten Verarbeitungsschritt von dem Speicher auf der einen Seite durch das Datenverarbeitungsfeld zum Speicher auf der anderen Seite strömen, dort die beim ersten Felddurchströmen erhaltenen Zwischenergebnisse im zweiten Speicher abgelegt werden, gegebenenfalls das Feld umkonfiguriert wird, die Zwischenergebnisse dann für

die Weiterverarbeitung zurückströmen usw., kann etwa eine Speicherseite durch eine LOAD-Konfiguration in einem Array-Teil mit neuen Daten vorgeladen werden, während aus der gegenüberliegenden Speicherseite Daten mit einer STORE-Konfiguration in einem anderen Array-Teil weggeschrieben werden. Dieses simultane LOAD/STORE-Vorgehen ist im übrigen auch ohne räumliche Speicherbereichstrennung möglich.

Es sei noch einmal erwähnt, dass es verschiedene Möglichkeiten gibt, interne Speicher mit Daten zu füllen. Die internen Speicher können insbesondere vorher durch separate Ladekonfigurationen unter Verwendung von datenstromartigem Zugreifen vorgeladen werden. Dies entsprichte dem Gebrauch als Vektorregister, wobei es zu Folge hat, dass die internen Speicher immer zumindest partiell ein Teil des nach außen sichtbaren Zustandes der XPP sein werden und daher bei Kontextwechseln gespeichert bzw. zurückgeschrieben werden müssen. Alternativ und/oder zusätzlich können die internen Speicher (iRAMs) durch separate „Lade-Instruktionen" auf die CPU geladen werden. Dies führt zu verringerten Ladevorgängen durch Konfigurationen und kann eine breitere Schnittstelle zur Speicherhierarchie bewirken. Wiederum wird wie auf Vektorregister zugegriffen.

Das Vorladen kann auch als Burst aus dem Speicher durch eine Anweisung des Cache-Kontrollers bestehen. Überdies ist es möglich, und dies ist als besonders leistungsfähig in vielen Fällen bevorzugt, den Cache dahingehend so auszubilden, dass eine bestimmte Vorladeanweisung eine bestimmte Speicherfläche, die durch die Startadresse und -größe bzw. Schrittweite (n) definiert ist, auf den internen Speicher (iRAM) abbildet. Wenn alle internen RAMs zugeordnet sind, kann die nächste

Konfiguration aktiviert werden. Die Aktivierung bringt ein Abwarten mit sich, bis alle burst-artigen Ladevorgänge abgeschlossen sind. Dies ist jedoch insoweit transparent, wenn die Vorladeanweisungen lange genug vorher ausgegeben werden und die Cache-Lokalisierung nicht durch Interrupts oder Taskwechsel zerstört wird. Es kann dann insbesondere eine „Preload clean"-Anweisung verwendet werden, mit der vermieden wird, dass Daten aus dem Speicher geladen werden.

Eine Synchronisierungsanweisung wird benötigt, um sicherzustellen, dass der Inhalt eines spezifischen Speicherbereiches, der cacheartig im IRAM abgelegt ist, an die Speicherhierarchie zurückgeschrieben werden kann, was global oder durch Spezifizierung des Speicherbereiches, auf welchen zugegriffen wird, erfolgen kann; der globale Zugriff entspricht einem „full write back". Um das Vorladen des IRAMs zu vereinfachen, ist es möglich, dies durch einfache Angabe einer Basisadresse, ggf. einer bzw. mehrerer Schrittweiten (beim Zugriff auf multidimensionale Datenfelder) sowie einer Gesamtlauflänge zu spezifizieren und diese in Registern oder dergleichen abzulegen und dann, zur Bestimmung, wie geladen werden soll, auf diese Register zuzugreifen.

Besonders bevorzugt ist es, wenn die Register als FIFOs ausgebildet werden. Es kann dann für eine Vielzahl auch virtueller Prozessoren in einer Multithreadumgebung jeweils ein FIFO vorgesehen werden. Überdies können Speicherstellen zum Gebrauch als TAG-Speicher wie bei Caches üblich vorgesehen werden.

Es sei auch erwähnt, dass die Markierung des Inhaltes von IRAMs als im Cache-Sinne „dirty" hilfreich ist, damit der In-

halt so schnell wie möglich an einen externen Speicher zurückgeschrieben werden kann, wenn er nicht im gleichen IRAM wieder verwendet werden soll. Damit können das XPP-Feld und der Cache-Controller als eine einzige Einheit betrachtet werden, da sie keine unterschiedlichen Anweisungsströme benötigen. Vielmehr kann der Cache-Controller als die Implementierung der Stufen „configuration fetch", „operand fetch" (IRAM preload) und „write back", also CF, OF und WB, in der XPP-Pipeline angesehen werden, wobei auch die Ausführungsstufe (ex) ausgelöst wird. Auf Grund der langen Latenzen und der Nichtvorhersehbarkeit etwa durch Cache-Fehlzugriffe oder Konfigurationen mit unterschiedlicher Länge ist es vorteilhaft, wenn die Stufen mehrere Konfigurationen breit überlappt werden, wobei zwecks loser Ankopplung das Konfigurations- und Datenvorlade-FIFO (Pipeline) verwendet wird. Es sei erwähnt, dass dem Preload der per se bekannte FILMO nachgeordnet sein kann. Es sei auch erwähnt, dass das Vorladen spekulativ sein kann, wobei das Spekulationsmaß compilerabhängig bestimmt werden kann. Ein Nachteil durch falsches Vorladen entsteht aber insofern nicht, als nicht ausgeführte, sondern nur vorgeladene Konfigurationen ohne weiteres für ein Überschreiben freigebbar sind, genauso wie zugeordnete Daten. Die Vorladung des FIFOs kann mehrere Konfigurationen weit vorausgehen und etwa abhängig von Eigenschaften des Algorithmus sein. Es ist möglich, eine Hardware hierfür zu verwenden.

Was das Zurückschreiben von verwendeten Daten aus dem IRAM in externe Speicher angeht, so kann dies durch einen geeigneten, der XPP zugeordneten Cache-Controller erfolgen, wobei aber darauf hingewiesen wird, dass dieser typisch seine Aufgaben priorisieren wird und bevorzugt Vorladeoperationen ausführt, die auf Grund des zugeordneten Ausführungsstatusses eine hohe

Priorität besitzen. Andererseits kann auch ein Vorladen durch eine überlagerte IRAM-Instanz in einem anderen Block oder den Mangel an leeren IRAM-Instanzen im Ziel-IRAM-Block blockiert werden. In letzerem Fall kann die Konfiguration warten, bis eine Konfiguration und/oder ein Zurückschreiben beendet ist. Die IRAM-Instanz in einem unterschiedlichen Block kann dabei im Gebrauch befindlich oder „dirty" sein. Es kann vorgesehen werden, dass die zuletzt verwendeten *sauberen* IRAMs verworfen werden, also als „leer" betrachtet werden. Wenn weder leere noch saubere IRAM-Instanzen vorliegen, muss ein „dirty"-IRAM-Teil bzw. ein nichtleerer an die Speicherhierarchie zurückgeschrieben werden. Da sich immer nur eine Instanz im Gebrauch befinden kann, und es mehr als eine Instanz in einem IRAM-Block geben soll, damit ein Cache-Effekt erreicht wird, kann es nicht passieren, dass weder leere noch saubere noch „dirty"-IRAM-Instanzen existieren.

Beispiele von Architekturen, bei denen ein SMT-Prozessor mit einer XPP-Thread-Resource gekoppelt ist, finden sich beispielhaft in Fig. 4a - c.

Auch bei der hier vorgestellten und bevorzugten Variante ist es erforderlich, ggf. den Speicherverkehr zu beschränken, was während Kontextwechseln auf verschiedene Weisen möglich ist. So brauchen reine Lesedaten nicht gespeichert werden, wie dies etwa bei Konfigurationen der Fall ist. Bei nicht unterbrechbaren (nicht präemptiven) Konfigurationen brauchen die lokalen Zustände von Bussen und PAEs nicht gespeichert werden.

Es kann vorgesehen werden, dass nur modifizierte Daten gespeichert werden und es können Cache-Strategien verwendet

werden, um den Speicherverkehr zu verringern. Hierzu kann insbesondere bei häufigen Kontextwechseln eine LRU-Strategie (LRU = least recently used) insbesondere zusätzlich zu einem Vorlademechanismus implementiert werden.

Wenn IRAMs als lokale Cache-Kopien des Hauptspeichers definiert werden und jedem IRAM eine Startadresse und Modifizierungszustandsinformation zugeordnet ist, ist es bevorzugt, dass die IRAM-Zellen auch wie für die SMT-Unterstützung repliziert sind, so dass nur die Startadressen der IRAMs gespeichert und als Kontext wieder geladen werden müssen. Die Startadressen für die IRAMs einer augenblicklichen Konfiguration wählen dann die IRAM-Instanzen mit identischen Adressen zum Gebrauch aus. Wenn kein Adress-TAG einer IRAM-Instanz der Adresse des neu geladenen bzw. neu zu ladenden Kontexts entspricht, kann der entsprechende Speicherbereich in eine leere IRAM-Instanz geladen werden, wobei dies hier zu verstehen ist als freier IRAM-Bereich. Wenn ein solcher nicht verfügbar ist, kann auf die vorbeschriebenen Verfahren zurückgegriffen werden.

Es sei im übrigen darauf hingewiesen, dass durch Rückschreiben bedingte Verzögerungen unter Verwendung einer insbesondere separaten Statemachine (Cache-Controller) vermieden werden können, mit welcher versucht wird, momentan inaktive IRAM-Instanzen während nicht benötigter Speicherzyklen zurückzuschreiben.

Es sei darauf hingewiesen, dass, wie aus dem oben Stehenden ersichtlich ist, bevorzugt der Cache als expliziter Cache und nicht als wie gewöhnlich dem Programmierer und/oder Compiler transparenter Cache aufzufassen ist. Um hier die entsprechen-

de Ansteuerung vorzusehen, können, etwa durch den Compiler, folgende Anweisungen ausgegeben werden: Konfigurationsvorladeanweisungen, die IRAM-Vorladeanweisungen vorausgehen, welche von jener Konfiguration verwendet werden. Derartige Konfigurationsvorladeinstruktionen sollten so früh wie möglich vom Scheduler vorgesehen werden. Weiter, das heißt alternativ und/oder zusätzlich, können IRAM-Vorladeinstruktionen vorgesehen werden, die gleichfalls frühzeitig vom Scheduler vorgesehen werden sollten, und es können Konfigurationsausführungsanweisungen vorgesehen werden, die IRAM-Vorladeanweisungen für diese Konfiguration folgen, wobei diese Konfigurationsausführungsanweisungen insbesondere um abgeschätzte Latenzzeiten gegenüber den Vorladeanweisungen verzögern können.

Es kann auch vorgesehen werden, dass eine Konfigurationswarteanweisung ausgeführt wird, gefolgt von einer Anweisung, die ein Cache-Zurückschreiben erzwingt, wobei beides vom Compiler ausgegeben wird, und zwar insbesondere dann, wenn eine Anweisung einer anderen funktionellen Einheit wie der Lade-/Speichereinheit auf einen Speicherbereich zugreifen kann, der potenziell „dirty" oder in einem IRAM in Gebrauch befindlich ist. Damit kann eine Synchronisierung der Anweisungsströme und der Cache-Inhalte unter Vermeidung von Daten-Hazards erzwungen werden. Durch entsprechende Handhabung sind derartige Synchronisationsanweisungen nicht notwendigerweise häufig.

Es sei erwähnt, dass das Datenladen und/oder -ablegen nicht zwingend durch vollständig logikzellenfeldbasiertes Vorgehen erfolgen muss. Vielmehr ist es auch möglich, etwa eine oder mehrere separate und/oder dedizierte DMA-Einheiten, das heißt insbesondere DMA-Controller vorzusehen, die z. B. allenfalls

noch durch Vorgaben bezüglich Startadresse, Schrittweite, Blockgröße, Zieladressen etc. insbesondere von der CT und/ oder aus dem Logikzellenfeld konfiguriert bzw. funktionsvorbereitet und/oder eingerichtet werden.

Das Laden auch kann insbesondere aus einem Cache und in diesen hinein erfolgen. Dies hat die Vorteile, dass die externe Kommunikation mit größeren Speicherbänken über den Cachecontroller gehandhabt wird, ohne dass innerhalb des Datenverarbeitungslogikzellenfeldes separate Schaltanordnungen dafür vorgesehen sein müssen, dass der Zugriff in lesender oder schreibender Weise bei Cache-Speichermitteln typisch sehr schnell und mit allenfalls geringer Latenzzeit erfolgen wird und dass auch typisch eine CPU-Einheit, dort typisch über eine separate LOAD/STORE-Einheit, an diesen Cache angebunden ist, sodass ein Zugriff auf Daten und ein Austausch derselben zwischen CPU-Kern und Datenverarbeitungslogikzellenfeld blockweise schnell und derart erfolgen kann, dass nicht für jedes Übergeben von Daten ein separater Befehl etwa aus dem OpCode-Fetcher der CPU abgeholt und verarbeitet werden muss.

Es erweist sich diese Cacheankoppelung auch als wesentlich günstiger als eine Ankopplung eines Datenverarbeitungslogikzellenfeldes an die ALU über Register, wenn diese Register nur über eine LOAD/STORE-Einheit mit einem Cache kommunizieren, wie dies aus den Nicht-PACT-eigenen zitierten Schriften per se bekannt ist.

Es kann eine weitere Datenverbindung zu der Lade/Speichereinheit der oder einer dem Datenverarbeitungslogikzellenfeld zugeordneten Sequenziell-CPU-Einheit vorgesehen sein und/oder zu deren Register.

Es sei erwähnt, dass ein Ansprechen derartiger Einheiten über separate Eingangs-/Ausgangsanschlüsse (IO-Ports) der insbesondere als VPU beziehungsweise XPP ausgestaltbaren Datenverarbeitungslogikzellenanordnung erfolgen kann und/oder durch einen oder mehrere einem Einzelport nachgeschaltete Multiplexer.

Dass neben dem insbesondere blockweisen und/oder streamenden und/oder im Random-Access, insbesondere im RMW-Modus (Read-Modify-Write-Modus) erfolgenden Zugriff auf Cache-Bereiche in schreibender und/oder lesender Weise und/oder die LOAD/STORE-Einheit und/oder die (per se im Stand der Technik bekannte) Verbindung mit dem Register der Sequenziell-CPU auch eine Verbindung mit einem externen Massenspeicher wie einem RAM, einer Festplatte und/oder einem anderen Datenaustauschport wie einer Antenne und so weiter erfolgen kann, sei auch erwähnt. Es kann für diesen Zugriff auf Cache- und/oder LOAD/STORE-Einheit- und/oder registereinheitverschiedene Speichermittel ein separater Port vorgesehen sein. Dass hier geeignete Treiber, Buffer, Signalaufbereiter für Pegelanpassung und so weiter vorgesehen sein können, z. B. LS 74244, LS74245, sei erwähnt. Im Übrigen sei erwähnt, dass insbesondere, jedoch nicht ausschließlich zur Aufbereitung eines in das Datenverarbeitungslogikzellenfeld hineinströmenden oder in diesem strömenden Datenstromes die Logikzellen des Feldes ALUs bzw. EALUs umfassen können und typisch werden, denen eingangs- und/oder ausgangsseitig, insbesondere sowohl eingangs- als auch ausgangsseitig kurze, feingranular konfigurierbare, FPGA-artige Schaltkreise vorgesetzt sein können und/oder in die PAE-ALU integriert sein können, um etwa aus einem kontinuierlichen Datenstrom Bitblöcke herauszuschneiden, wie dies

für die MPEG-4-Dekodierung erforderlich ist. Es ist dies zum einen vorteilhaft, wenn ein Datenstrom in die Zelle hineingelangen soll und dort ohne Blockierung von größeren PAE-Einheiten einer Art Vorverarbeitung zu unterwerfen ist. Dies ist auch dann von ganz besonderem Vorteil, wenn die ALU als SIMD-Rechenwerk ausgestaltet wird, wobei dann ein sehr breites Dateneingangswort von zum Beispiel 32 Bit Datenbreite über die vorgeschalteten FPGA-artigen Streifen aufgespalten wird in mehrere parallele Datenwörter von zum Beispiel 4 Bit Breite, die dann in den SIMD-Rechenwerken parallel abgearbeitet werden können, was die Gesamtperformance des Systems signifikant zu erhöhen vermag, sofern entsprechende Anwendung benötigt werden. Es sei darauf hingewiesen, dass vorstehend von FPGA-artigen vor- beziehungsweise nachgeschalteten Strukturen die Rede war. Mit FPGA-artig muss aber, was explizit erwähnt sei, nicht zwingend Bezug genommen sein auf 1-Bit-granulare Anordnungen. Es ist insbesondere möglich, statt dieser hyperfeingranularen Strukturen lediglich feiner granulare Strukturen von zum Beispiel 4 Bit Breite vorzusehen. Das heißt, die FPGA-artigen Eingangs- und/oder Ausgangsstrukturen vor und/oder nach einer insbesondere als SIMD-Rechenwerk ausgestalteten ALU-Einheit sind z.B. so konfigurierbar, dass immer 4 Bit breite Datenwörter zugeführt und/oder verarbeitet werden. Es ist möglich, hier eine Kaskadierung vorzusehen, so dass zum Beispiel die einkommenden 32 Bit breiten Datenwörter in 4 separierte bzw. separierende 8-Bit-FPGA-artige, nebeneinander angeordnete Strukturen strömen, diesen 4 Stück 8 Bit breiten FPGA-artigen Strukturen ein zweiter Streifen mit 8 Stück 4 Bit breiten FPGA-artigen Strukturen nachgesetzt ist, und gegebenenfalls nach einem weiteren derartigen Streifen dann, sofern dies für den jeweiligen Zweck als erforderlich erachtet wird, zum Beispiel 16 Stück parallel nebeneinander

angeordnete 2 Bit breite FPGA-artige Strukturen vorgesehen werden. Wenn dies der Fall ist, kann gegenüber rein hyperfeingranular FPGA-artigen Strukturen eine beträchtliche Verringerung des Konfigurationsaufwandes erzielt werden. Dass dies überdies dazu führt, dass der Konfigurationsspeicher und so weiter der FPGA-artigen Struktur wesentlich kleiner ausfallen kann und somit eine Einsparung an Chipfläche erzielt wird, sei erwähnt. Auch sei erwähnt, dass FPGA-artige Streifenstrukturen wie auch in Verbindung mit Fig. 3 offenbart, insbesondere bezüglich Anordnung in der PAE, besonders leicht die Implementierung von Pseudozufallsrauschgeneratoren ermöglichen. Wenn dabei schrittweise immer wieder aus einer einzigen FPGA-Zelle erhaltene einzelne Ausgangsbits an die FPGA-Zelle zurückgespeichert werden, kann auch mit einer einzigen Zelle sequenziell ein Pseudozufallsrauschen kreativ generiert werden, was als für sich erfinderisch betrachtet wird, vgl. Fig. 5.

Prinzipiell sind die vorstehend beschriebenen Kopplungsvorteile bei Datenblockströmen über den Cache erreichbar; besonders bevorzugt ist es jedoch, wenn der Cache streifenweise (slice-artig) aufgebaut ist und dann ein Zugriff auf mehrere der Slices simultan erfolgen kann, insbesondere auf alle Slices gleichzeitig. Dies ist dann vorteilhaft, wenn, was noch erörtert werden wird, auf dem Datenverarbeitungslogikzellenfeld (XPP) und/oder der Sequenziell-CPU und/oder den Sequenziell-CPUs eine Vielzahl von Threads abzuarbeiten sind, sei es im Wege des Hyperthreadings, des Multitaskings und/oder des Multithreadings. Es sind also bevorzugt Cachespeichermittel mit Scheibenzugriff bzw. Scheibenzugriffsermöglichungssteuermitteln vorgesehen. Es kann dabei z. B. jedem Thread eine eigene Scheibe zugeordnet werden. Dies ermöglicht es

später, beim Abarbeiten der Threads sicherzustellen, dass jeweils auf die entsprechenden Cachebereiche bei Wiederaufnahme der mit dem Thread abzuarbeitenden Befehlsgruppe zugegriffen wird.

Es sei noch einmal erwähnt, dass der Cache nicht zwingend in Slices unterteilt sein muss, und dass, wenn dies der Fall ist, nicht zwingend jeder Slice einem eigenen Thread zugewiesen werden muss. Es sei allerdings darauf hingewiesen, dass dies die bei weitem bevorzugte Methode ist. Es sei weiter darauf hingewiesen, dass es Fälle geben kann, in denen nicht alle Cache-Bereiche simultan oder zu einer gegebenen Zeit temporär benützt werden. Vielmehr ist zu erwarten, dass bei typischen Datenverarbeitungsanwendungen, wie sie in handgehaltenen mobilen Telefonen (Handys) , Laptops, Kameras und so weiter auftreten werden, häufig Zeiten vorliegen werden, in denen nicht der gesamte Cache benötigt wird. Es ist daher besonders bevorzugt, wenn einzelne Cache-Bereiche von der Leistungsversorgung derart trennbar sind, dass ihr Energieverbrauch signifikant absinkt, insbesondere auf oder nahe null. Dies kann bei sliceweiser Ausgestaltung des Caches durch sliceweise Abschaltung derselben über geeignete Leistungsabtrennmittel geschehen, vgl. zum Beispiel Fig. 2. Die Abtrennung kann entweder über eine Heruntertaktung, Taktabtrennung oder eine Leistungsabtrennung erfolgen. Es kann insbesondere einer einzelnen Cache-Scheibe oder dergleichen eine Zugriffserkennung zugeordnet sein, welche dazu ausgebildet ist, zu erkennen, ob ein jeweiliger Cache-Bereich beziehungsweise eine jeweilige Cache-Scheibe momentan einen ihm zugeordneten Thread, Hyperthread oder Task hat, von welchem er benützt wird. Sofern dann vom Zugriffserkennungsmittel festgestellt wird, dass dies nicht der Fall ist, wird typisch eine Abtren-

nung vom Takt und/oder sogar der Leistung möglich sein. Es sei erwähnt, dass bei Wiedereinschalten der Leistung nach einem Abtrennen ein sofortiges Wiederansprechen des Cachebereiches möglich ist, also keine signifikante Verzögerung durch das An- und Ausschalten der Leistungszufuhr zu erwarten ist, sofern mit gängigen geeigneten Halbleitertechnologien eine Implementierung in Hardware erfolgt. Dies ist unabhängig von der Verwendung mit Logikzellenfeldern in vielen Anwendungen sinnvoll.

Ein weiterer besonderer Vorteil, der sich bei der vorliegenden Erfindung ergibt, besteht darin, dass zwar eine besonders effiziente Kopplung bezüglich des Übertrags von Daten beziehungsweise Operanden in insbesondere blockweiser Form gegeben ist, dass aber dennoch ein Balancing nicht in der Weise erforderlich ist, dass die exakt gleiche Verarbeitungszeit in Sequenziell-CPU und XPP beziehungsweise Datenverarbeitungslogikzellenfeld erforderlich ist. Vielmehr erfolgt die Verarbeitung in einer praktisch oftmals unabhängigen Weise, insbesondere derart, dass die Sequenziell-CPU und die Datenverarbeitungslogikzellenfeldanordnung für einen Scheduler oder dergleichen als separate Ressourcen betrachtbar sind. Dies erlaubt eine sofortige Umsetzung bekannter Datenverarbeitungsprogrammaufspaltungstechnologien wie Multitasking, Multithreading und Hyperthreading. Der sich ergebende Vorteil, dass ein Pfadbalancing nicht erforderlich ist, das heißt Ausbalancieren zwischen sequenziellen Teilen (z. B. auf einer RISC-Einheit) und Datenflussteilen (z. B. auf einer XPP) führt dazu, dass beispielsweise innerhalb der Sequenziell-CPU (also z. B. den RISC functional units) beliebige Anzahlen von Pipelinestufen durchlaufen werden können, Taktungen in unterschiedlicher Weise möglich sind und so weiter. Ein weiterer

Vorteil der vorliegenden Erfindung besteht darin, dass durch das Hineinkonfigurieren einer Ladekonfiguration beziehungsweise einer Storekonfiguration in die XPP oder andere Datenverarbeitungslogikzellenfelder die Daten in das Feld mit einer Geschwindigkeit hineingeladen werden oder aus diesem herausgeschrieben werden können, die nicht mehr bestimmt ist durch die Taktgeschwindigkeit der CPU, die Geschwindigkeit, mit welcher der OpCode-Fetcher arbeitet, oder dergleichen. Mit anderen Worten ist die Ablaufsteuerung der Sequenziell-CPU nicht mehr flaschenhalsartig begrenzend für den Datendurchsatz des Datenzellenlogikfeldes, ohne dass eine nur noch lose Ankopplung besteht.

Während es in einer besonders bevorzugten Variante der Erfindung möglich ist, die für eine XPP-Einheit bekannte CT (bzw. CM; Konfigurationsmanager bzw. Konfigurationstabelle) zu verwenden, um sowohl das Konfigurieren eines oder mehrerer, auch hierarchisch mit mehreren CTs angeordneter XPP-Felder und gleichzeitig eines oder mehrerer Sequenziell-CPUs, dort quasi als Multithreading-Scheduler- und -Hardwareverwaltung zu verwenden, was den inhärenten Vorteil hat, daß bekannte Technologien wie z. B. FILMO usw. für die hardwareunterstützte Verwaltung beim Multithreading einsetzbar werden, ist es alternativ und/oder, insbesondere in hierarchischer Anordnung, zusätzlich möglich, dass ein Datenverarbeitungslogikzellenfeld wie eine XPP Konfigurationen vom OpCode-Fetcher einer Sequenziell-CPU über das Koprozessor-Interface erhält. Dies führt dazu, daß von der Sequenziell-CPU und/oder einer anderen XPP ein Aufruf instantiiert werden kann, der zu einer Datenabarbeitung auf der XPP führt. Die XPP wird dabei dann z. B. über die beschriebene Cache-Ankopplung und/oder mittels LOAD- und/oder STORE-Konfigurationen, die Adressgeneratoren für La-

den und/oder Wegschreiben von Daten im XPP- bzw. Datenverarbeitungslogikzellenfeld vorsehen, im Datenaustausch gehalten. Mit anderen Worten wird eine coprozessorartige und/oder Thread-Ressourcen-artige Ankopplung eines Datenverarbeitungslogikzellenfeldes möglich, während gleichzeitig ein datenstromartiges Datenladen durch Cache- und/oder I/O-Port-Kopplung erfolgt.

Es sei erwähnt, daß die Koprozessor-Ankopplung, d. h. die Ankopplung des Datenverarbeitungslogikzellenfeldes typisch dazu führen wird, daß das Scheduling auch für dieses Logikzellenfeld auf der Sequenziell-CPU oder einer dieser übergeordneten Schedulereinheit bzw. einem entsprechenden Schedulermittel erfolgen wird. In einem solchen Fall findet praktisch die Threading-Kontrolle und -verwaltung auf dem Scheduler bzw. der Sequenziell-CPU statt. Obwohl dies per se möglich ist, wird dies, zumindest bei einfachster Implementierung der Erfindung, nicht zwingend der Fall sein. Vielmehr kann eine Verwendung des Datenverarbeitungslogikzellenfeldes durch Aufruf in herkömmlicher Weise wie bei einem Standard-Coprozessor, etwa bei 8086/8087-Kombinationen erfolgen.

Weiter sei erwähnt, daß es in einer besonders bevorzugten Variante, unabhängig von der Art der Konfiguration, sei es über das Coprozessor-Interface, den als Scheduler mitdienenden Konfigurationsmanager (CT) der XPP bzw. des Datenverarbeitungslogikzellenfeldes oder dergleichen oder auf andere Weise, möglich ist, im bzw. unmittelbar am Datenverarbeitungslogikzellenfeld bzw. unter Verwaltung des Datenverarbeitungslogikzellenfeldes Speicher, insbesondere interne Speicher, insbesondere bei der XPP-Architektur, wie sie aus den diversen Voranmeldungen und den Veröffentlichungen des Anmelders be-

kannt ist, RAM-PAEs, oder andere entsprechend verwaltete oder interne Speicher wie ein Vektorregister anzusprechen, d. h. die über die LOAD-Konfiguration eingeladenen Datenmengen vektorartig wie in Vektorregistern in die internen Speicher abzulegen, dann, nach Umkonfigurieren der XPP bzw. des Datenverarbeitungslogikzellenfeldes, also Überschreiben bzw. Nachladen und/oder Aktivieren einer neuen Konfiguration, die die eigentliche Verarbeitung der Daten durchführt (in diesem Zusammenhang sei darauf hingewiesen, daß für eine solche Verarbeitungskonfiguration auch Bezug genommen werden kann auf eine Mehrzahl von Konfigurationen, die z. B. im Wave-Modus und/oder sequenziell nacheinander abzuarbeiten sind) zuzugreifen wie bei einem Vektorregister und dann die dabei erhaltenen Ergebnisse und/oder Zwischenergebnisse wiederum in die internen oder über die XPP wie interne Speicher verwalteten externen Speicher, um dort diese Ergebnisse abzulegen. Die so vektorregisterartig mit Verarbeitungsergebnissen beschriebenen Speichermittel unter XPP-Zugriff sind dann, nach Rekonfigurieren der Verarbeitungskonfiguration durch Laden der STORE-Konfiguration in geeigneter Weise weggeschrieben, was wiederum datenstromartig geschieht, sei es über den I/O-Port direkt in externe Speicherbereiche und/oder, wie besonders bevorzugt, in Cache-Speicherbereiche, auf welche dann zu einem späteren Zeitpunkt die Sequenziell-CPU und/oder andere Konfigurationen auf der zuvor die Daten erzeugt habenden XPP oder einer anderen entsprechenden Datenverarbeitungseinheit zugreifen können.

Eine besonders bevorzugte Variante besteht darin, zumindest für bestimmte Datenverarbeitungsergebnisse und/oder Zwischenergebnisse als Speicher- bzw. Vektorregistermittel, in welchem bzw. welches die erhaltenen Daten abzulegen sind, nicht

einen internen Speicher zu benutzen, in welchen Daten über eine STORE-Konfiguration in den Cache- oder einen anderen Bereich, auf welchen die Sequenziell-CPU oder eine andere Datenverarbeitungseinheit zugreifen können, wegzuschreiben sind, sondern statt dessen unmittelbar die Ergebnisse wegzuschreiben in entsprechende, insbesondere zugriffsreservierte Cachebereiche, die insbesondere Slice-artig organisiert sein können. Dies kann gegebenenfalls den Nachteil einer größeren Latenz haben, insbesondere, wenn die Wege zwischen der XPP- oder Datenverarbeitungslogikzellenfeldeinheit und dem Cache so lang sind, daß die Signallaufzeiten ins Gewicht fallen, führt aber dazu, daß gegebenenfalls keine weitere STORE-Konfiguration benötigt wird. Es sei im übrigen erwähnt, daß eine derartige Abspeicherung von Daten in Cache-Bereiche einerseits, wie vorstehend beschrieben, dadurch möglich ist, daß der Speicher, in welchen geschrieben wird, physikalisch nahe beim Cache-Controller liegt und als Cache ausgestaltet ist, dass aber alternativ und/oder zusätzlich auch die Möglichkeit besteht, einen Teil eines XPP-Speicherbereiches, XPP-internen Speichers oder dergleichen, insbesondere bei RAM über PAEs, vgl. PACT31 (DE 102 12 621.6, WO 03/036507), unter die Verwaltung eines oder, nacheinander mehrerer Cache-Speichercontroller zu stellen. Dies hat dann Vorteile, wenn die Latenz beim Abspeichern der Verarbeitungsergebnisse, welche innerhalb des Datenverarbeitungslogikzellenfeldes bestimmt werden, gering gehalten werden soll, während die Latenz beim Zugriff auf den dann nur noch als „Quasi-Cache" dienenden Speicherbereich durch andere Einheiten nicht oder nicht signifikant ins Gewicht fällt.

Es sei im übrigen erwähnt, daß auch eine Ausgestaltung derart möglich ist, daß der Cache-Controller einer herkömmlichen Se-

quenziell-CPU einen Speicherbereich als Cache anspricht, der, ohne dem Datenaustausch mit dem Datenverarbeitungslogikzellenfeld zu dienen, auf und/oder bei diesem physikalisch liegt. Dies hat den Vorteil, daß dann, wenn Anwendungen auf dem Datenverarbeitungslogikzellenfeld laufen, die einen allenfalls geringen lokalen Speicherbedarf haben, und/oder wenn auch nur wenige weitere Konfigurationen bezogen auf die zur Verfügung stehenden Speichermengen benötigt werden, diese einer oder mehreren Sequenziell-CPUs als Cache zur Verfügung stehen können. Es sei erwähnt, daß dann der Cache-Controller für die Verwaltung eines Cache-Bereiches mit dynamischem Umfang, d. h. variierender Größe ausgebildet sein kann und wird. Eine dynamische Cache-Umfangsverwaltung bzw. Cache-Umfangsverwaltungsmittel für die dynamische Cache-Verwaltung wird typisch die Arbeitslast und/oder die Input-/Output-Last auf der Sequenziell-CPU und/oder dem Datenverarbeitungslogikzellenfeld berücksichtigen. Mit anderen Worten kann beispielsweise analysiert werden, wie viele NOPs Datenzugriffe in einer gegebenen Zeiteinheit auf der Sequenziell-CPU vorliegen und/oder wie viele Konfigurationen im XPP-Feld in dafür vorgesehenen Speicherbereichen vorabgelegt sein sollen, um eine schnelle Umkonfiguration, sei es im Wege einer Wellenrekonfiguration oder auf andere Weise ermöglichen zu können. Die hiermit offenbarte dynamische Cachegrösse ist dabei insbesondere bevorzugt laufzeitdynamisch, d. h . der Cachecontroller verwaltet jeweils eine aktuelle Cachegrösse, die sich von Takt zu Takt oder Taktgruppe ändern kann. Es sei im übrigen darauf hingewiesen, daß die Zugriffsverwaltung eines XPP- bzw. Datenverarbeitungslogikzellenfeldes mit Zugriff als interner Speicher wie bei einem Vektorregister und als Cacheartiger Speicher für den externen Zugriff was die Speicherzugriffe angeht bereits beschrieben wurde in der DE 196 54 595

und der PCT/DE 97/03013 (PACT03). Die genannten Schriften sind durch Bezugnahme zu Offenbarungszwecken hiermit vollumfänglich eingegliedert.

Vorstehend wurde auf Datenverarbeitungslogikzellenfelder Bezug genommen, die insbesondere zur Laufzeit rekonfigurierbar sind. Es wurde diskutiert, dass bei diesen eine Konfigurationsverwaltungseinheit (CT bzw. CM) vorgesehen werden kann. Aus den diversen, zu Offenbarungszwecken unter Bezug genommenen Schutzrechten des Anmelders sowie seinen weiteren Veröffentlichungen ist die Verwaltung von Konfigurationen per se bekannt. Es sei nun explizit darauf hingewiesen, dass derartige Einheiten und deren Wirkungsweise, mit der insbesondere unabhängig von Ankopplungen an Sequenziell-CPUs etc. aktuell noch nicht benötigte Konfigurationen vorladbar sind, auch sehr gut nutzbar sind, um im Multitaskingbetrieb und/oder bei Hyperthreading und/oder Multithreading einen Task- beziehungsweise einen Thread- und/oder Hyperthreadwechsel zu bewirken, vgl. zum Beispiel 6a - 6c. Dazu kann ausgenützt werden, dass während der Laufzeit eines Threads oder Tasks in die Konfigurationsspeicher bei einer einzelnen oder einer Gruppe von Zellen des Datenverarbeitungslogikzellenfeldes, also beispielsweise einer PAE eines PAE-Feldes (PA) auch Konfigurationen für unterschiedliche Aufgaben, das heißt Tasks oder Threads beziehungsweise Hyperthreads geladen werden können. Dies führt dann dazu, dass bei einer Blockade eines Tasks oder Threads, etwa wenn auf Daten gewartet werden muss, weil diese noch nicht verfügbar sind, sei es, da sie von einer anderen Einheit noch nicht generiert oder empfangen wurden, beispielsweise auf Grund von Latenzen, sei es, weil eine Resource derzeit noch durch einen anderen Zugriff blockiert ist, dann Konfigurationen für einen anderen Task oder Thread

vorladbar und/oder vorgeladen sind und auf diese gewechselt werden kann, ohne dass der
Zeitoverhead für einen Konfigurationswechsel bei der insbesondere schattengeladenen Konfiguration abgewartet werden muss. Während es prinzipiell möglich ist, diese Technik auch dann zu verwenden, wenn innerhalb eines Tasks die wahrscheinlichste Weiterführung vorhergesagt wird und eine Vorhersage nicht zutrifft (prediction miss), wird diese Art des Betriebs bei vorhersagefreiem Betrieb bevorzugt sein. Bei Verwendung mit einer rein sequentiellen CPU und/oder mehreren rein sequentiellen CPUs, insbesondere ausschließlich mit solchen, wird somit durch die Zuschaltung eines Konfigurationsmanagers eine Multithreadingverwaltungshardware realisiert. Verwiesen sei hinsichtlich dessen insbesondere auf PACT10 (DE 198 07 872.2, WO 99/44147, WO 99/44120) und PACT17 (DE 100 28 397.7, WO 02/13000). Dabei kann es als ausreichend erachtet werden, insbesondere dann, wenn nur für eine CPU und/oder einige wenige Sequenziell-CPUs eine Hyperthreadingverwaltung gewünscht ist, auf bestimmte, in den speziell unter Bezug genommenen Schutzrechten beschriebene Teilschaltungen wie den FILMO zu verzichten. Insbesondere wird damit die Verwendung der dort beschriebenen Konfigurationsmanager mit und/oder ohne FILMO für die Hyperthreadingverwaltung für eine und/oder mehrere rein sequenziell arbeitende CPUs mit oder ohne Ankopplung an eine XPP oder ein anderes Datenverarbeitungslogikzellenfeld offenbart und hiermit für sich beansprucht. Es wird hierin eine für sich erfinderische Besonderheit gesehen. Es sei im Übrigen erwähnt, dass eine Vielzahl von CPUs realisiert werden kann mit den bekannten Techniken, wie sie insbesondere aus PACT31 (DE 102 12 621.6-53, PCT/EP 02/10572) und PACT34 (DE 102 41 812.8, PCT/EP 03/09957) bekannt sind, bei welchen innerhalb eines Arrays eine oder mehrere Sequenziell-CPUs

aufgebaut werden unter Ausnutzung eines oder mehrerer Speicherbereiche insbesondere im Datenverarbeitungslogikzellenfeld für den Aufbau der sequenziellen CPU, insbesondere als Befehls- und/oder Datenregister. Auch sei darauf verwiesen, dass bereits in früheren Anmeldungen wie PACT02, (DE 196 51 075.9-53, WO 98/26356), PACT04 (DE 196 54 846.2-53, WO 98/29952), PACT08, (DE 197 04 728.9, WO 98/35299) offenbart wurde, wie Sequenzer mit Ring- und/oder Wahlfrei-Zugriff-Speichern aufgebaut werden können.

Es sei darauf hingewiesen, dass ein Task- beziehungsweise Thread- und/oder Hyperthreadwechsel unter Verwendung der bekannten CT-Technologie, vgl. PACT10 (DE 198 07 872.2, WO 99/44147, WO 99/44120) und PACT17 (DE 100 28 397.7, WO 02/13000) derart erfolgen kann und bevorzugt auch erfolgen wird, dass einem per se bekannten, Software-implementierten Betriebssystem-Scheduler oder dergleichen von der CT Performance-Scheiben und/oder Zeitscheiben zugeordnet werden, während welchen bestimmt wird, von welchen Tasks oder Threads nachfolgend welche Teile per se, unterstellt, dass Resourcen frei sind, abzuarbeiten sind. Dazu sei ein Beispiel wie folgt gegeben: Zunächst soll für einen ersten Task eine Adressfolge generiert werden, gemäß welcher während der Ausführung einer LOAD-Konfiguration Daten aus einem Speicher und/oder Cache-Speicher, an dem ein Datenverarbeitungslogikzellenfeld in der beschriebenen Weise angekoppelt ist, geladen werden sollen. Sobald diese Daten vorliegen, kann mit der Abarbeitung einer zweiten, der eigentlichen Datenverarbeitungskonfiguration, begonnen werden. Auch diese kann vorgeladen werden, da sicher feststeht, dass diese Konfiguration, sofern keine Interrupts oder dergleichen einen vollständigen Taskwechsel erzwingen, auszuführen ist. In herkömmlichen Prozessoren ist nun das

Problem des sogenannten Cache-Miss bekannt, bei dem die Daten zwar angefordert werden, aber nicht im Cache für den Ladezugriff bereit liegen. Tritt ein solcher Fall in einer Kopplung gemäß der vorliegenden Erfindung auf, kann bevorzugt auf einen anderen Thread, Hyperthread und/oder Task gewechselt werden, der insbesondere zuvor von dem insbesondere softwareimplementierten Betriebssystem-Scheduler und/oder einer anderen hard- und/oder softwareimplementierten, entsprechend wirkenden Einheit für eine nächstmögliche Ausführung bestimmt wurde und demgemäß bevorzugt vorab in einen der verfügbaren Konfigurationsspeicher des Datenverarbeitungslogikzellenfeldes insbesondere im Hintergrund während der Ausführung einer anderen Konfiguration, beispielsweise der LOAD-Konfiguration, welche das Laden jener Daten, auf die nun gewartet wird, bewirkt hat, geladen wurde. Dass für die Vorabkonfiguration ungestört von der tatsächlichen Verschaltung der insbesondere grobgranular ausgebildeten Datenverarbeitungslogikzellen des Datenverarbeitungslogikzellenfeldes separate Konfigurationsleitungen von der konfigurierenden Einheit zu den jeweiligen Zellen direkt und/oder über geeignete Bussysteme geführt sein können wie per se im Stand der Technik bekannt, sei hier noch einmal explizit erwähnt, da diese Ausbildung hier besonders bevorzugt ist, um ein ungestörtes Vorabkonfigurieren ohne Störung einer anderen, gerade laufenden Konfiguration zu ermöglichen. Erwähnt seien hier u. a. die PACT10 (DE 198 07 872.2, WO 99/44147, WO 99/44120), PACT17 (DE 100 28 397.7, WO 02/13000) PACT13 (DE 199 26 538.0, WO 00/77652), PACT02 (DE 196 51 075.9, WO 98/26356) und PACT08 (DE 197 04 728.9, WO 98/35299). Wenn dann die Konfiguration, auf welche während beziehungsweise auf Grund des Task-Thread- und/oder Hyperthreadwechsels gewechselt wurde, abgearbeitet wurde, und zwar, bei bevorzugten nicht teilbaren, ununterbrechbaren und

somit quasi atomaren Konfigurationen bis zum Ende abgearbeitet wurde, vgl. PACT19 (DE 102 02 044.2, WO 2003/060747) und PACT11 (DE 101 39 170.6, WO 03/017095), wird teilweise eine weitere andere Konfiguration wie vorbestimmt durch die entsprechenden Scheduler, insbesondere den betriebssystemnahen Scheduler festgelegt, abgearbeitet und/oder jene Konfiguration, zu welcher zuvor die zugehörige LOAD-Konfiguration ausgeführt wurde. Vor der Ausführung einer Verarbeitungskonfiguration, zu welcher zuvor eine LOAD-Konfiguration ausgeführt wurde, kann insbesondere abgetestet werden, z. B. durch Abfrage des Status der LOAD-Konfiguration oder des datenladenden DMA-Kontrollers, ob mittlerweile die entsprechenden Daten in das Array eingeströmt sind, also die Latenzzeit, wie sie typisch auftritt, verstrichen ist und/oder die Daten tatsächlich vorliegen.

Mit anderen Worten werden dann Latenzzeiten, wenn sie auftreten, weil z. B. Konfigurationen noch nicht einkonfiguriert sind, Daten noch nicht geladen und/oder Daten noch nicht weggeschrieben wurden, überbrückt und/oder verdeckt, indem Threads, Hyperthreads und/oder Tasks ausgeführt werden, welche schon vorkonfiguriert sind und welche mit Daten arbeiten, die schon verfügbar sind beziehungsweise die an Ressourcen weggeschrieben werden können, die für das Wegschreiben bereits zur Verfügung stehen. Auf diese Weise werden Latenzzeiten weitgehend überdeckt und es wird, eine hinreichende Anzahl von per se auszuführenden Threads, Hyperthreads und/oder Tasks unterstellt, eine praktisch 100%-ige Ausnutzung des Datenverarbeitungslogikzellenfeldes erreicht.

Es sei besonders erwähnt, dass durch das Vorsehen hinreichend vieler XPP-interner Speicherresourcen, die frei - z. B. durch

den Scheduler oder die CT - Threads zugeordnet werden, zugleich und/oder überlagernd die Cache- und/oder Schreiboperationen mehrerer Threads durchgeführt werden können, was sich besonders positiv auf die Überbrückung eventueller Latenzen auswirkt.

Mit dem beschriebenen System bezüglich Datenstrom-Fähigkeit bei gleichzeitiger Ankopplung an eine Sequenziell-CPU und/oder bezüglich der Ankopplung eines XPP-Array beziehungsweise Datenverarbeitungslogikzellenfeldes und simultan einer Sequenziell-CPU an eine geeignete Schedulereinheit wie einen Konfigurationsmanager oder dergleichen lassen sich insbesondere ohne weiteres echtzeitfähige Systeme realisieren. Zur Echtzeitfähigkeit muss gewährleistet sein, dass auf eintreffende Daten beziehungsweise Interrupts, die insbesondere das Dateneintreffen signalisieren, innerhalb einer in keinem Fall zu überschreitenden Maximalzeit reagiert werden kann. Dies kann beispielsweise geschehen durch einen Taskwechsel auf einen Interrupt hin und/oder, beispielsweise bei priorisierten Interrupts, durch Festlegung, dass ein gegebener Interrupt momentan zu ignorieren ist, wobei auch dies innerhalb einer bestimmten Zeit festzulegen ist. Ein Taskwechsel bei derartigen echtzeitfähigen Systemen wird typisch auf drei Arten erfolgen können, nämlich entweder dann, wenn ein Task eine bestimmte Zeit gelaufen ist (Timer-Prinzip), bei Nichtzurverfügungstehen einer Resource, sei es durch deren Blockade durch anderen Zugriff oder aufgrund von Latenzen beim Zugriff darauf, insbesondere in schreibender und/oder lesender Weise, das heißt bei Latenzen von Datenzugriffen und/oder beim Auftreten von Interrupts.

Es wird im übrigen darauf hingewiesen, dass insbesondere auch eine laufzeitbegrenzte Konfiguration auf einer für die Interruptbearbeitung freizugebenden bzw. zu wechselnden Resource einen Watchdog bzw. Mitlaufzähler neu antriggern kann.

Während ansonsten explizit ausgeführt wurde, vgl. auch PACT 29 (DE 102 12 622.4, WO 03/081454), dass das Neuantriggern des Mitlaufzählers bzw. Watchdogs zur Laufzeiterhöhung durch einen Task-switch unterbindbar ist, wird vorliegend explizit offenbart, dass ein Interrupt gleichfalls, das heißt entsprechend einem Taskswitch, Mitlaufzähler - bzw. Watchdog - und Neutrigger blockierend wirken kann, d. h. es kann in einem solchen Fall unterbunden werden, dass die Konfiguration durch Neuantriggern selbst ihre maximal mögliche Laufzeit erhöht.

Mit der vorliegenden Erfindung kann die Echtzeitfähigkeit eines Datenverarbeitungslogikzellenfeldes nunmehr erreicht werden, indem eine oder mehrere von drei möglichen Varianten implementiert wird.

Eine erste Variante besteht darin, dass innerhalb einer von dem Scheduler beziehungsweise der CT ansprechbaren Ressource ein Wechsel zur Abarbeitung beispielsweise eines Interrupts erfolgt. Sofern die Ansprechzeiten auf Interrupts oder andere Anforderungen so groß sind, dass während dieser Zeit eine Konfiguration ohne Unterbrechung noch abgearbeitet werden kann, ist dies unkritisch, zumal während der Abarbeitung der aktuell laufenden Konfiguration auf jener Ressource, die für die Abarbeitung des Interrupts zu wechseln ist, eine Konfiguration zur Interruptabarbeitung vorgeladen werden kann. Die Auswahl der vorabzuladenden Interrupt-bearbeitenden Konfigu-

ration ist z. B. durch die CT durchzuführen. Es ist möglich, die Laufzeit der Konfiguration auf der für die Interruptbearbeitung freizugebenden bzw. zu wechselnden Ressource zu begrenzen. Verwiesen wird dazu auf PACT29/PCT(PCT/DE03/000942).

Bei Systemen, die schneller auf Interrupts reagieren müssen, kann es bevorzugt sein, eine einzelne Ressource, also beispielsweise eine separate XPP-Einheit und/oder Teile eines XPP-Feldes für eine solche Abarbeitung zu reservieren. Wenn dann ein schnell abzuarbeitender Interrupt auftritt, kann entweder eine für besonders kritische Interrupts schon vorab vorgeladene Konfiguration abgearbeitet werden oder es wird sofort mit dem Laden einer Interrupt behandelnden Konfiguration in die reservierte Ressource begonnen. Eine Auswahl der jeweils für den entsprechenden Interrupt erforderlichen Konfiguration ist durch entsprechende Triggerung, Waveabarbeitung usw. möglich.

Es sei im übrigen erwähnt, dass es mit den schon beschriebenen Methoden ohne weiteres möglich ist, eine instantane Reaktion auf einen Interrupt zu erhalten, indem über die Verwendung von LOAD/STORE-Konfigurationen eine Code-Reentranz erreicht wird. Hierbei wird nach jeder datenbearbeitenden Konfiguration oder zu gegebenen Zeiten, beispielsweise alle fünf oder zehn Konfigurationen eine STORE-Konfiguration ausgeführt und dann eine LOAD-Konfiguration unter Zugriff auf jene Speicherbereiche ausgeführt, in die zuvor weggeschrieben wurde. Wenn sichergestellt wird, dass die von der STORE-Konfiguration benutzten Speicherbereiche so lange unberührt bleiben, bis durch Fortschreiten im Task eine weitere Konfiguration sämtliche relevanten Informationen (Zustände, Daten) weggeschrieben hat, ist sichergestellt, dass bei Wiederladen, also

Wiedereintritt in eine zuvor bereits begonnene, aber nicht zu Ende geführte Konfiguration oder Konfigurationskette wieder dieselben Bedingungen erhalten werden. Eine solche Zwischenschaltung von LOAD/STORE-Konfigurationen unter simultanem Schutz von noch nicht veralteten STORE-Speicherbereichen lässt sich automatisch ohne zusätzlichen Programmieraufwand sehr einfach generieren, z. B. von einem Compiler. Dort kann die Ressourcenreservierung gegebenenfalls vorteilhaft sein. Das bei der Ressourcenreservierung und/oder in anderen Fällen auf zumindest eine Menge hochpriorisierter Interrupts durch Vorabladen von bestimmten Konfigurationen reagiert werden kann, sei noch einmal erwähnt.

Eine weitere, besonders bevorzugte Variante der Reaktion auf Interrupts besteht dann, wenn zumindest eine der ansprechbaren Ressourcen eine Sequenziell-CPU ist, darin, auf dieser eine Interrupt-Routine abzuarbeiten, in welcher wiederum Code für das Datenverarbeitungslogikzellenfeld verboten ist. Mit anderen Worten wird eine zeitkritische Interrupt-Routine ausschließlich auf einer Sequenziell-CPU abgearbeitet, ohne dass XPP-Datenverarbeitungsschritte aufgerufen werden. Dies garantiert, dass der Verarbeitungsvorgang auf dem Datenverarbeitungslogikzellenfeld nicht zu unterbrechen ist und es kann dann eine Weiterabarbeitung auf diesem Datenverarbeitungslogikzellenfeld nach einem Taskswitch erfolgen. Obwohl damit die eigentliche Interrupt-Routine keinen XPP-Code besitzt, kann dennoch dafür gesorgt werden, dass auf einen Interrupt hin zu einem späteren, nicht mehr echtzeitrelevanten Zeitpunkt mit der XPP auf einen durch einen Interrupt und/oder eine Echtzeitanforderung erfassten Zustand und/oder Daten unter Verwendung des Datenverarbeitungslogikzellenfeldes reagiert werden kann.

Patentansprüche

1. Datenverarbeitungsvorrichtung mit einem Datenverarbeitungslogikzellenfeld und zumindest einer Sequenziell-CPU, dadurch gekennzeichnet, dass eine Ankopplung der Sequenziell-CPU und des Datenverarbeitungslogikzellenfeldes zum Datenaustausch in insbesondere blockweiser Form durch zu einem Cache-Speicher führende Leitungen möglich ist.

2. Verfahren zum Betrieb einer rekonfigurierbaren Einheit mit Laufzeit beschränkten Konfigurationen, worin die Konfigurationen ihre maximal zulässige Laufzeit erhöhen können insbesondere durch Antriggern eines Mitlaufzählers, dadurch gekennzeichnet, dass eine Konfigurationslaufzeiterhöhung durch die Konfiguration im Ansprechen auf einen Interrupt unterbunden wird.

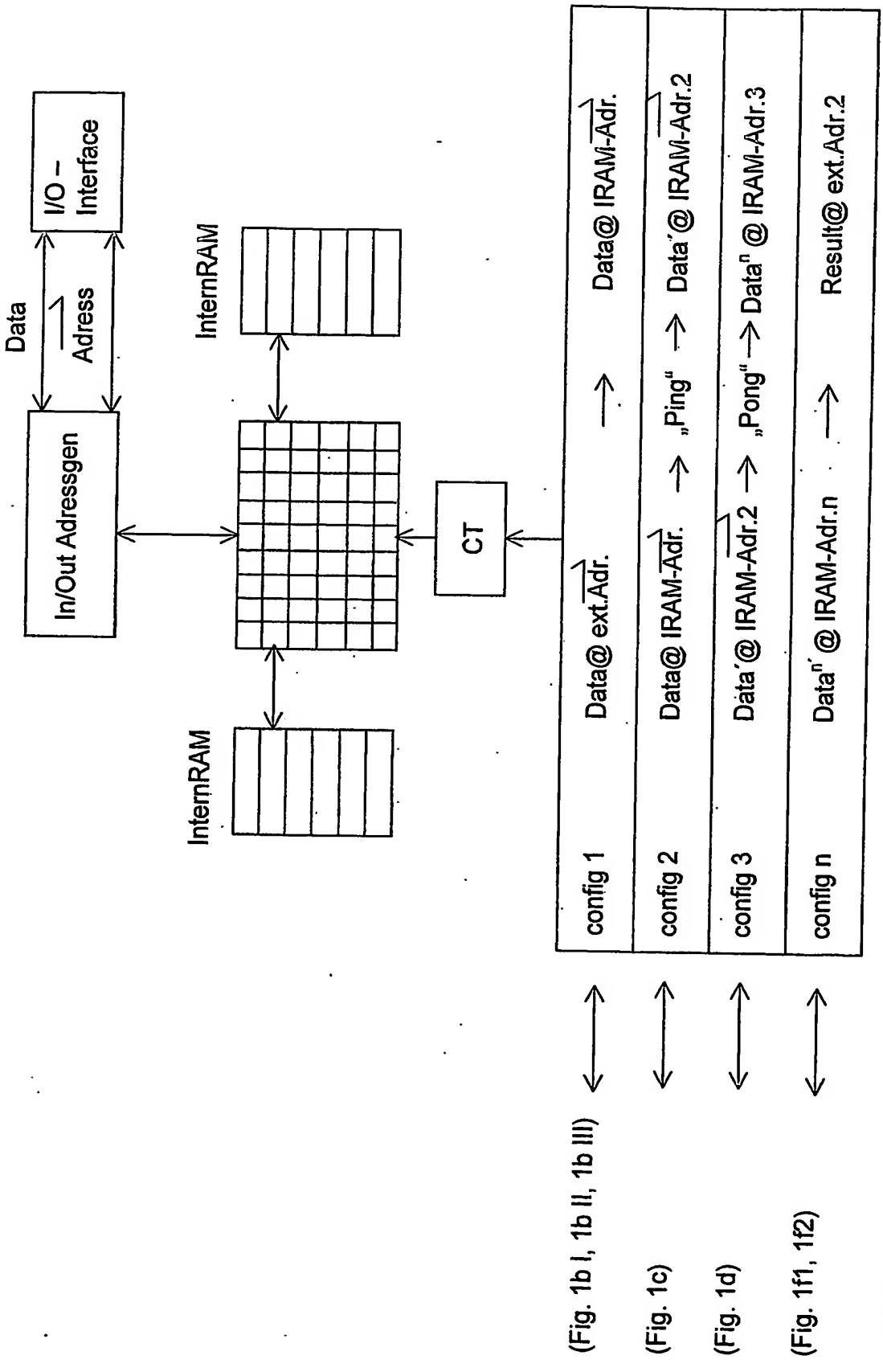

I/O – Interface
Data
Adress
InternRAM
In/Out Adressgen
CT
InternRAM
config 1
Data@ ext.Adr.
Data@ IRAM-Adr.
config 2
Data@ IRAM-Adr.
„Ping"
Data´@ IRAM-Adr.2
config 3
Data´@ IRAM-Adr.2
„Pong"
Data^n @ IRAM-Adr.3
config n
Data^n´ @ IRAM-Adr.n
Result@ ext.Adr.2
(Fig. 1b I, 1b II, 1b III)
(Fig. 1c)
(Fig. 1d)
(Fig. 1f1, 1f2)

Fig. 1a

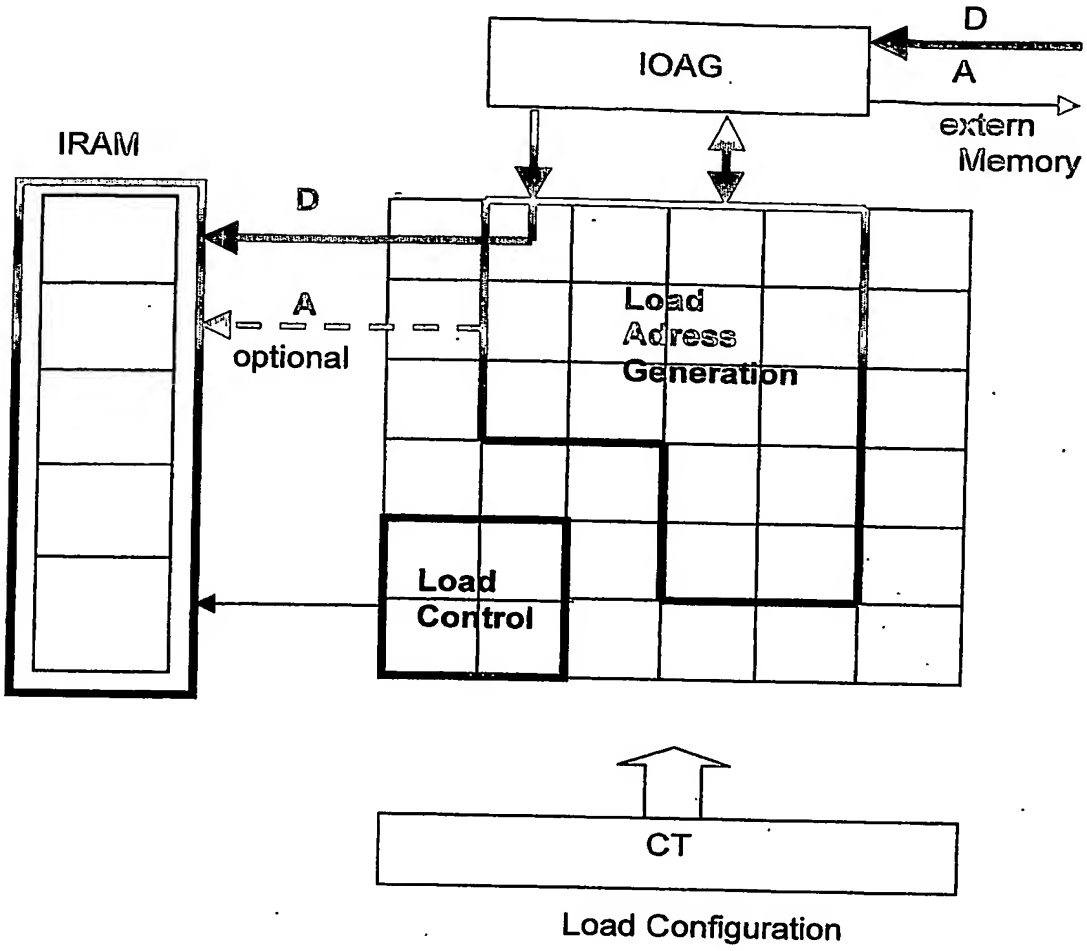

IOAG
D
A
extern
Memory
IRAM
D
A
optional
Load
Adress
Generation
Load
Control
CT
Load Configuration

Fig. 1b I

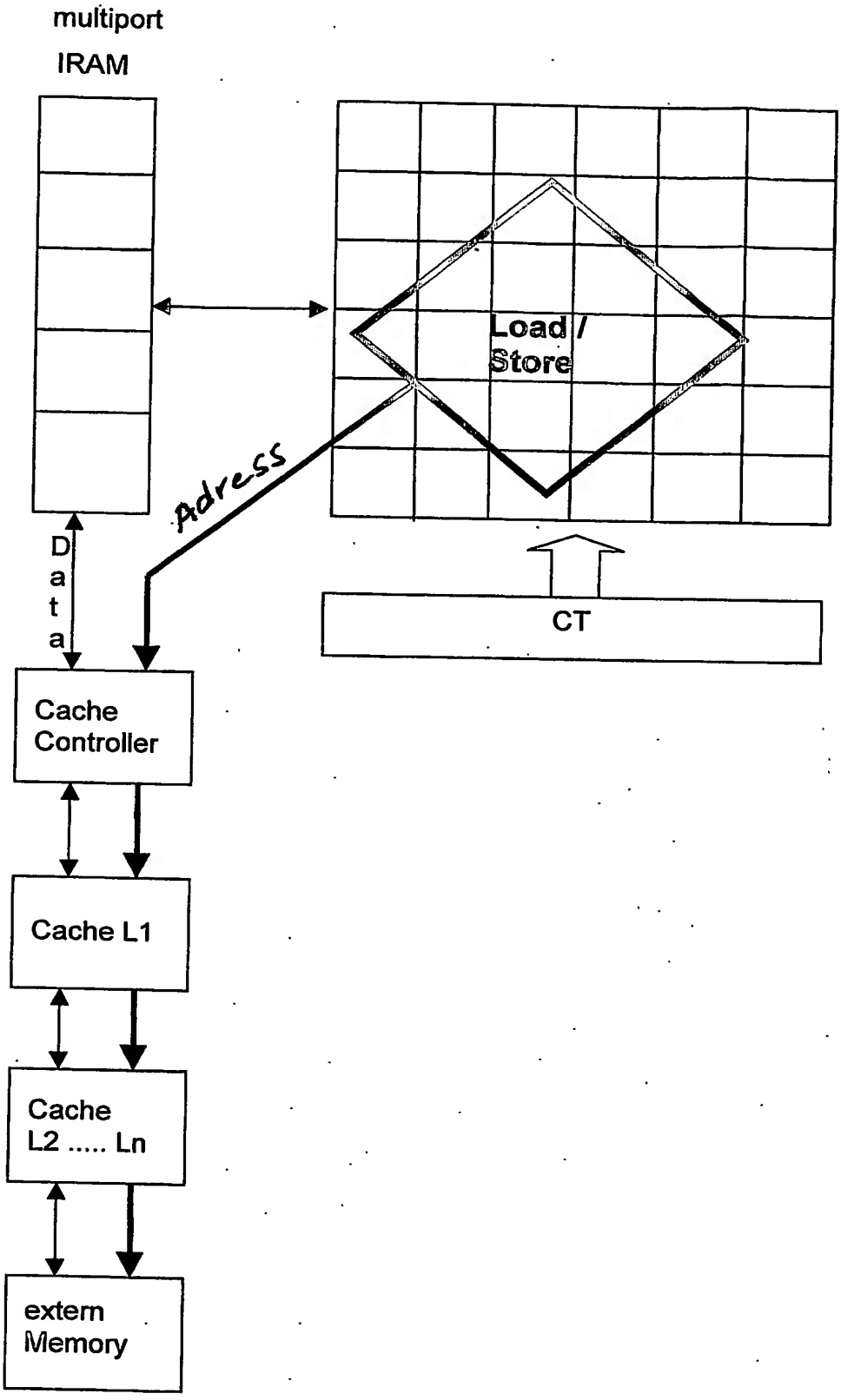

multiport
IRAM
Load /
Store
Adress
Data
CT
Cache
Controller
Cache L1
Cache
L2 ..... Ln
extern
Memory

Fig. 1b II

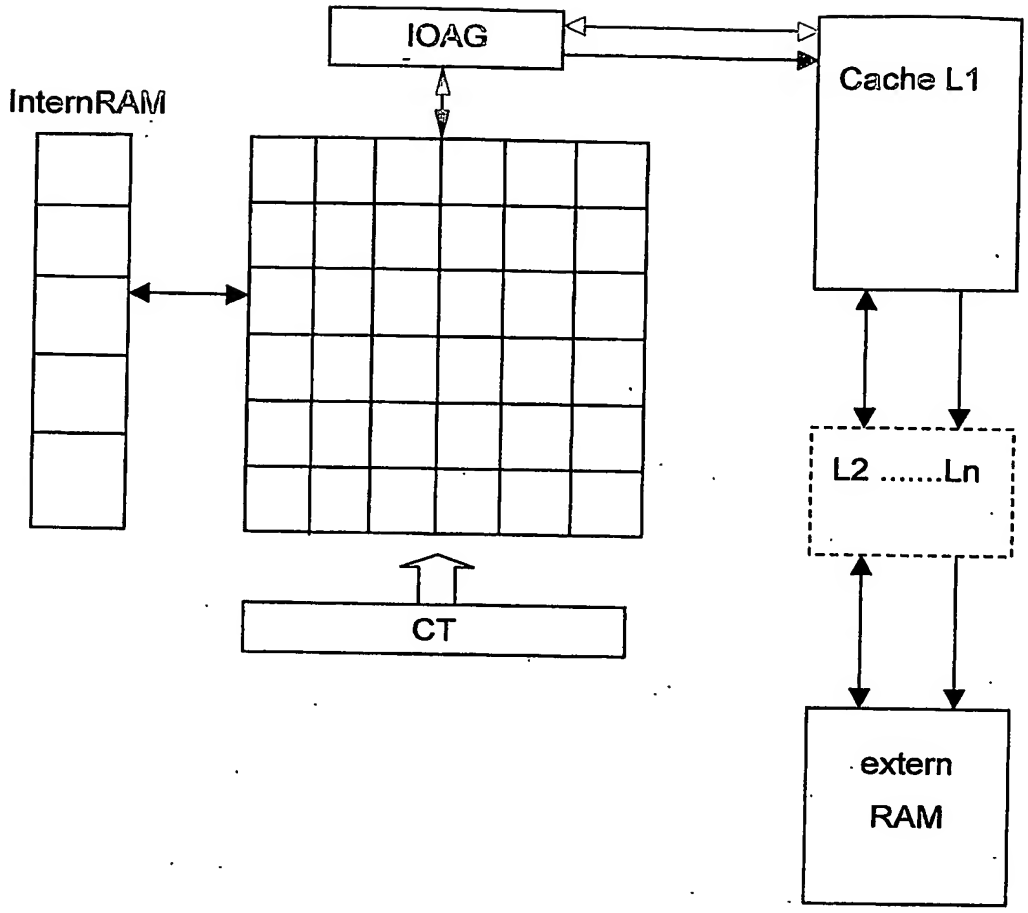


Fig. 1b III

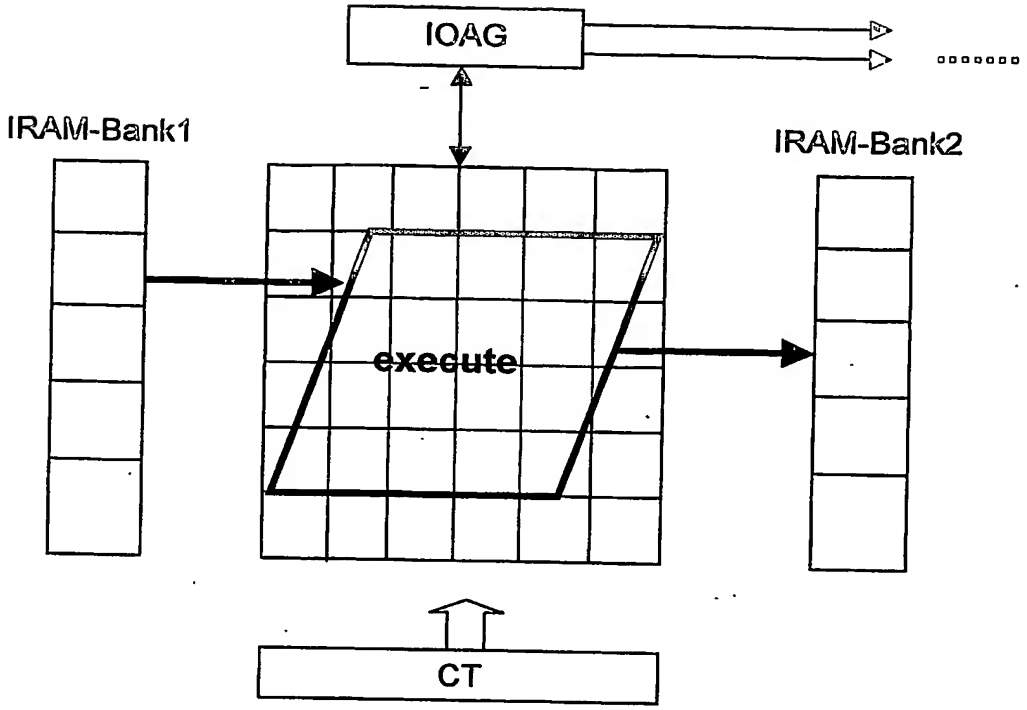


Fig. 1c

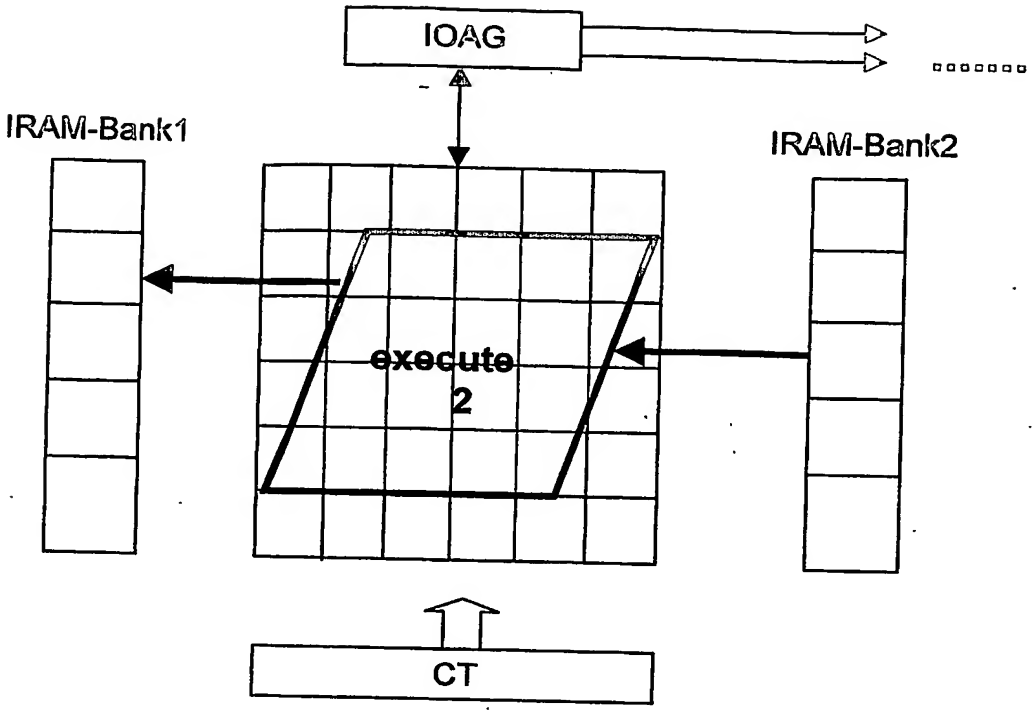
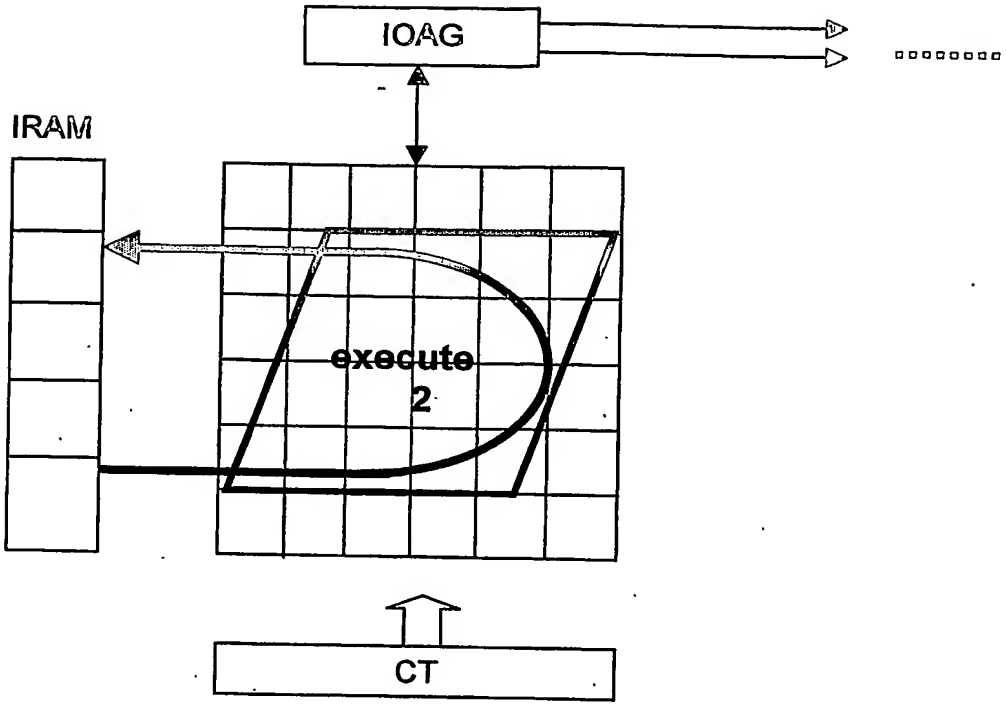


Fig. 1d



Ping / Pong

Fig. 1e1

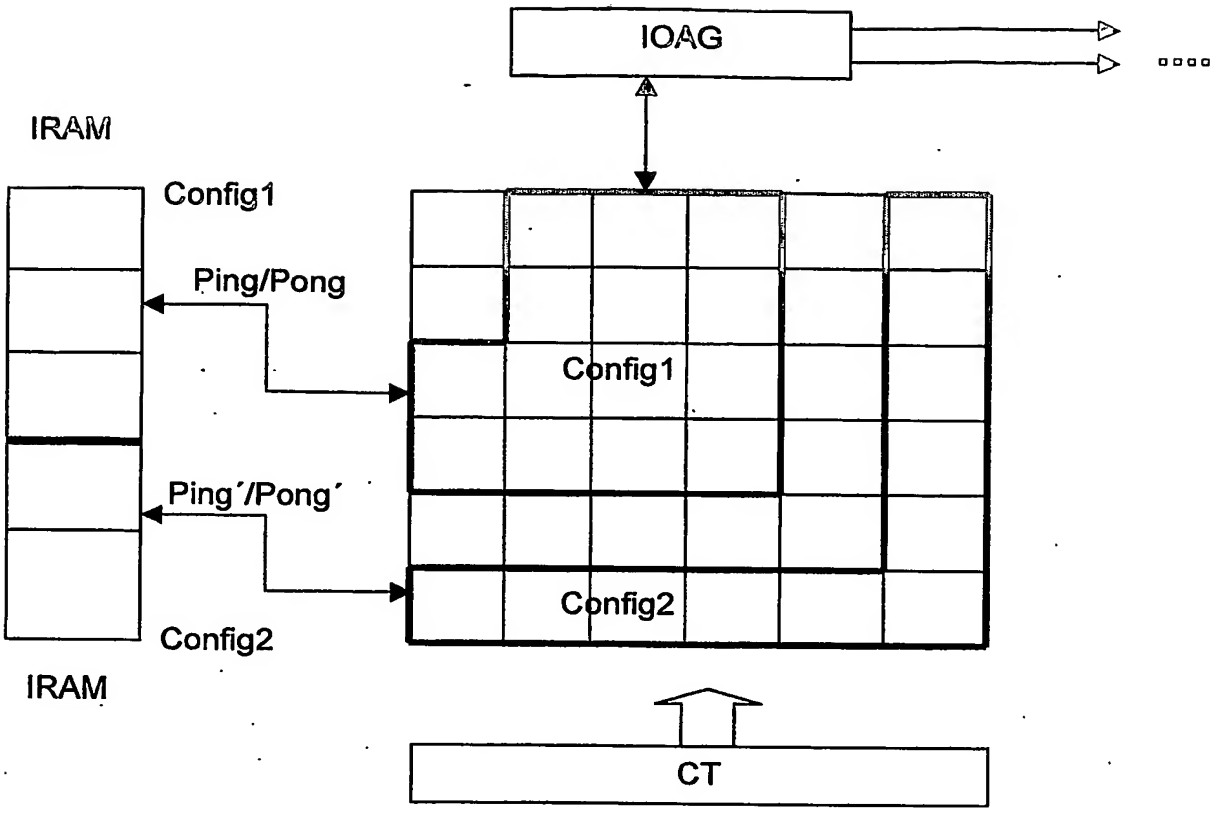

IOAG
IRAM
Config1
Ping/Pong
Config1
Ping´/Pong´
Config2
Config2
IRAM
CT

Fig. 1e2

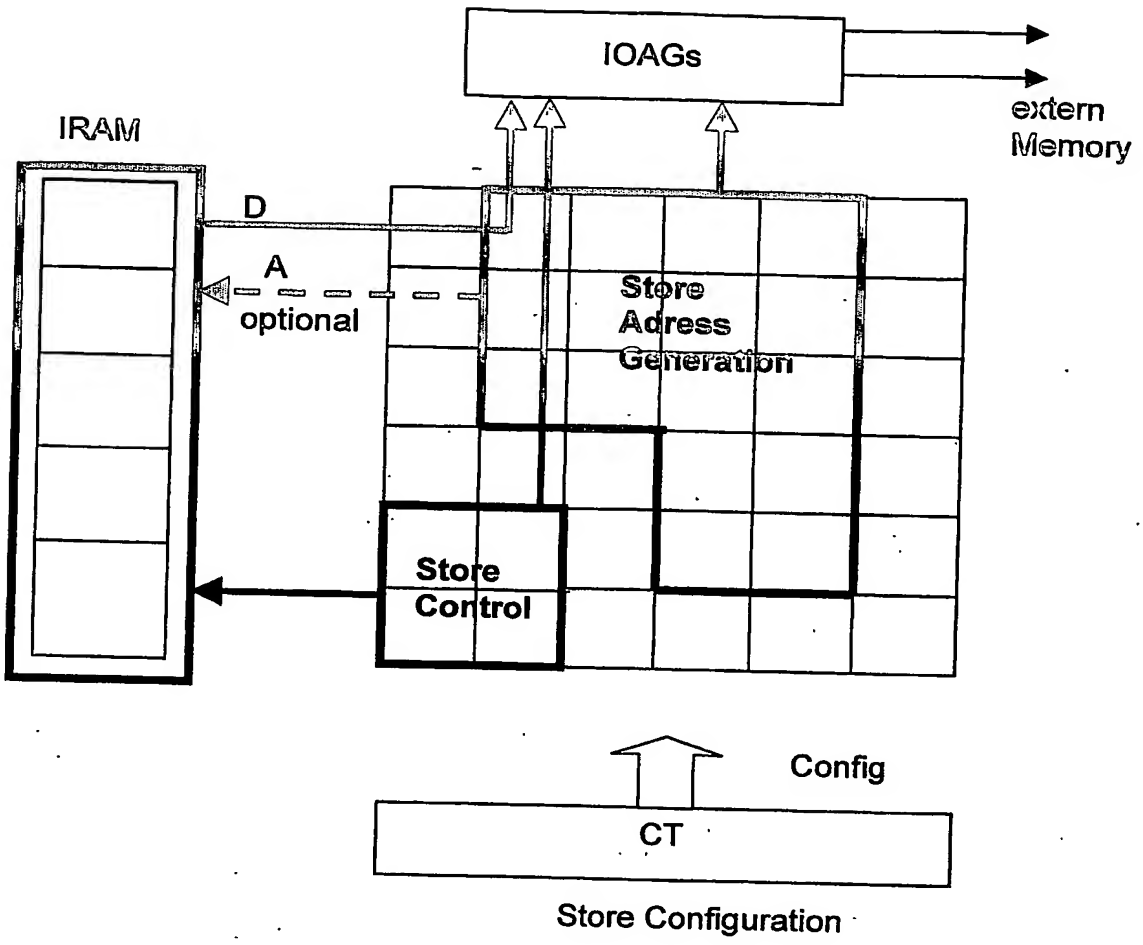

IOAGs
extern
Memory
IRAM
D
A
optional
Store
Adress
Generation
Store
Control
Config
CT
Store Configuration

Fig. 1f1

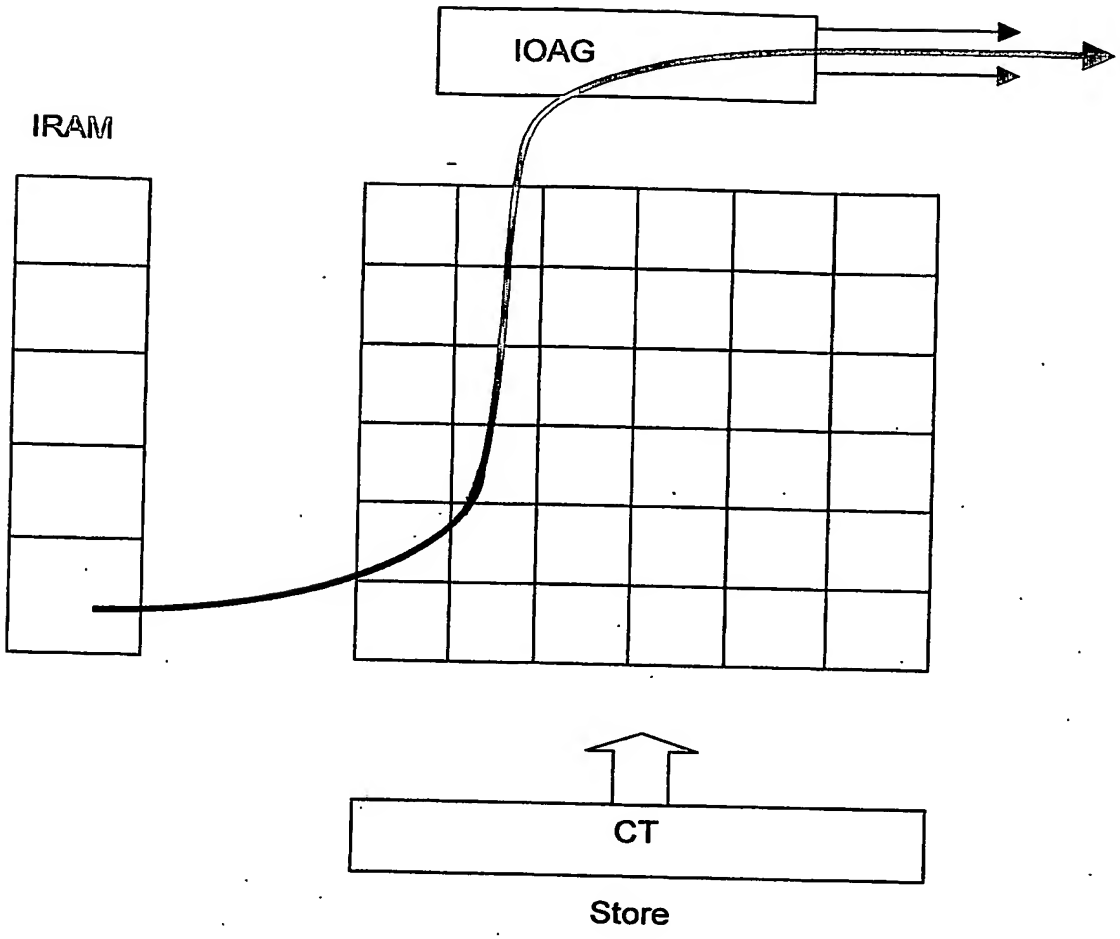


Fig. 1f2

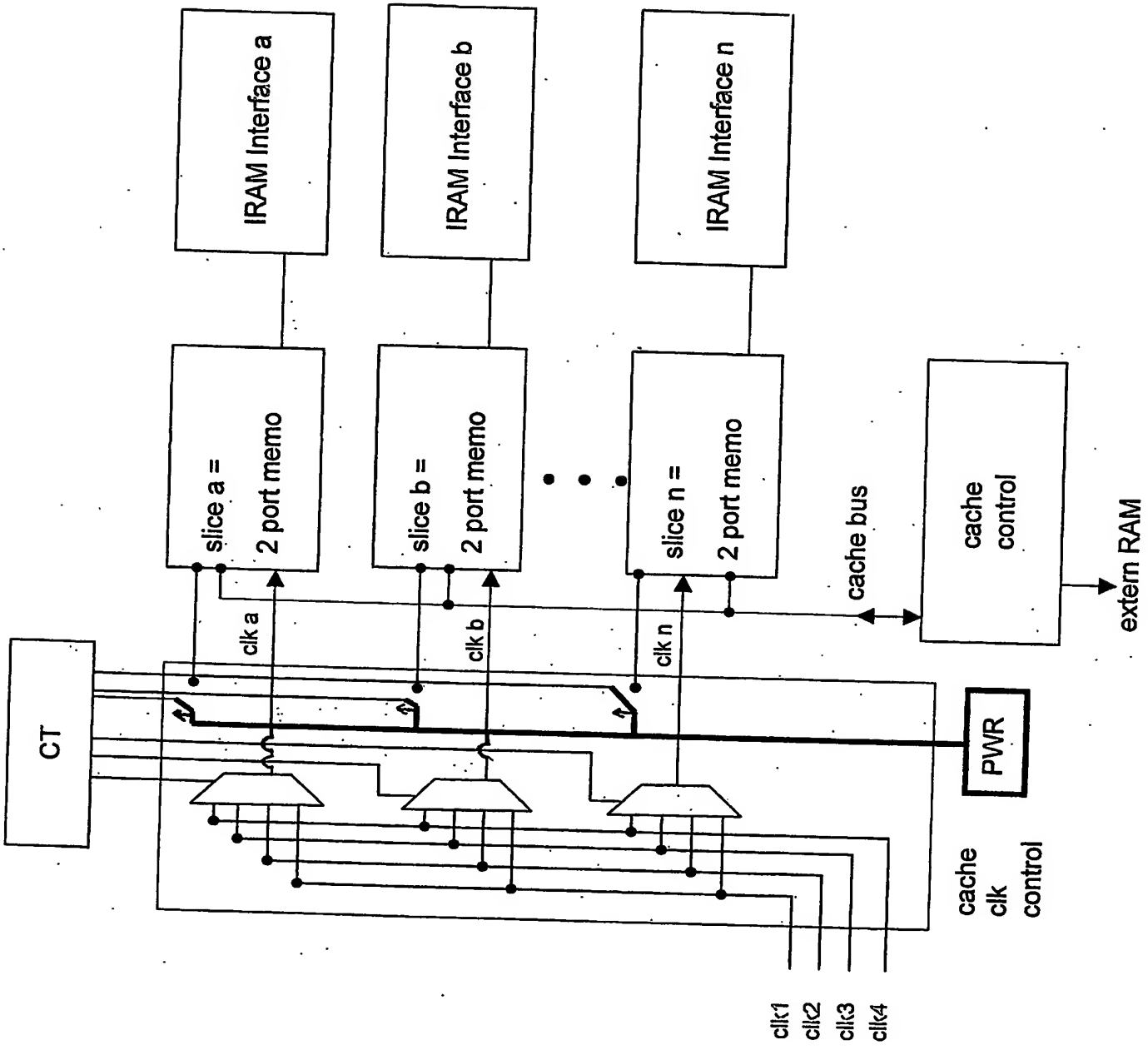

IRAM Interface a
IRAM Interface b
IRAM Interface n
slice a = 2 port memo
slice b = 2 port memo
slice n = 2 port memo
cache bus
cache control
extern RAM
CT
clk a
clk b
clk n
PWR
cache clk control
clk1
clk2
clk3
clk4

Fig. 2

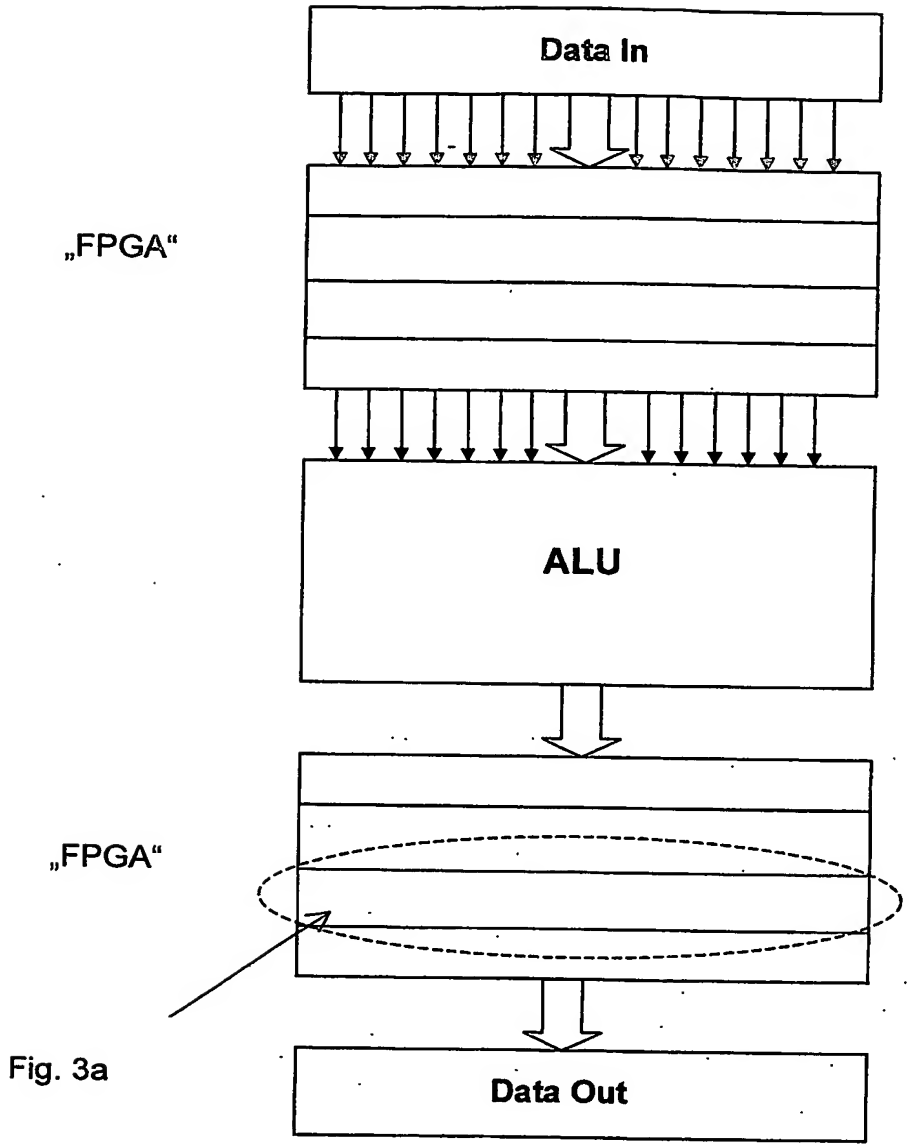


Fig. 3a

Fig. 3 I

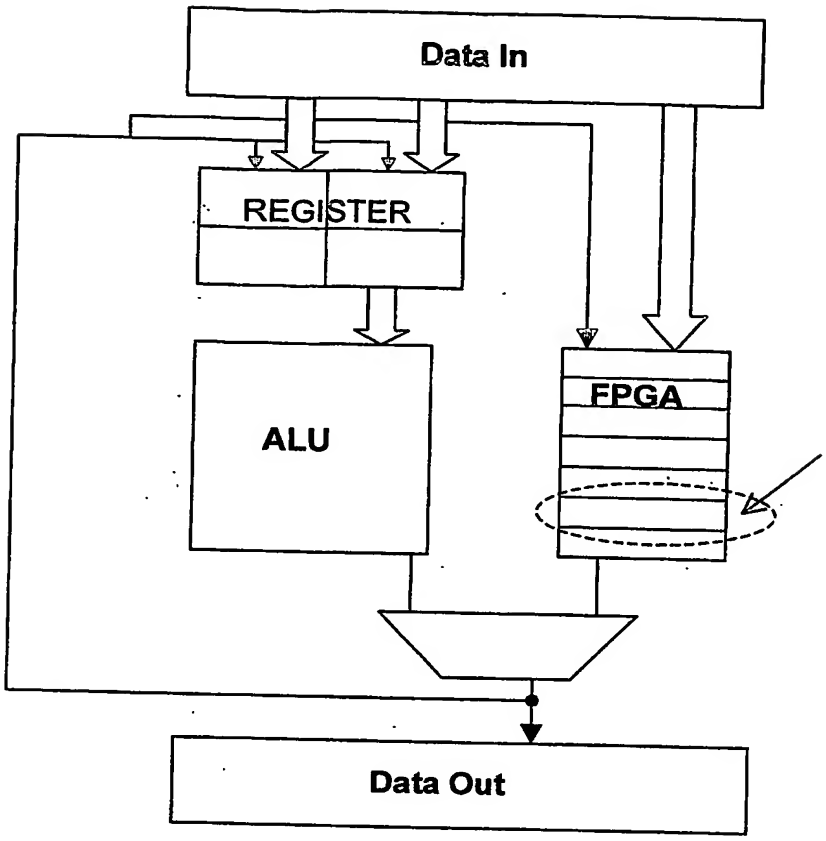


Fig. 3a

Fig. 3 II

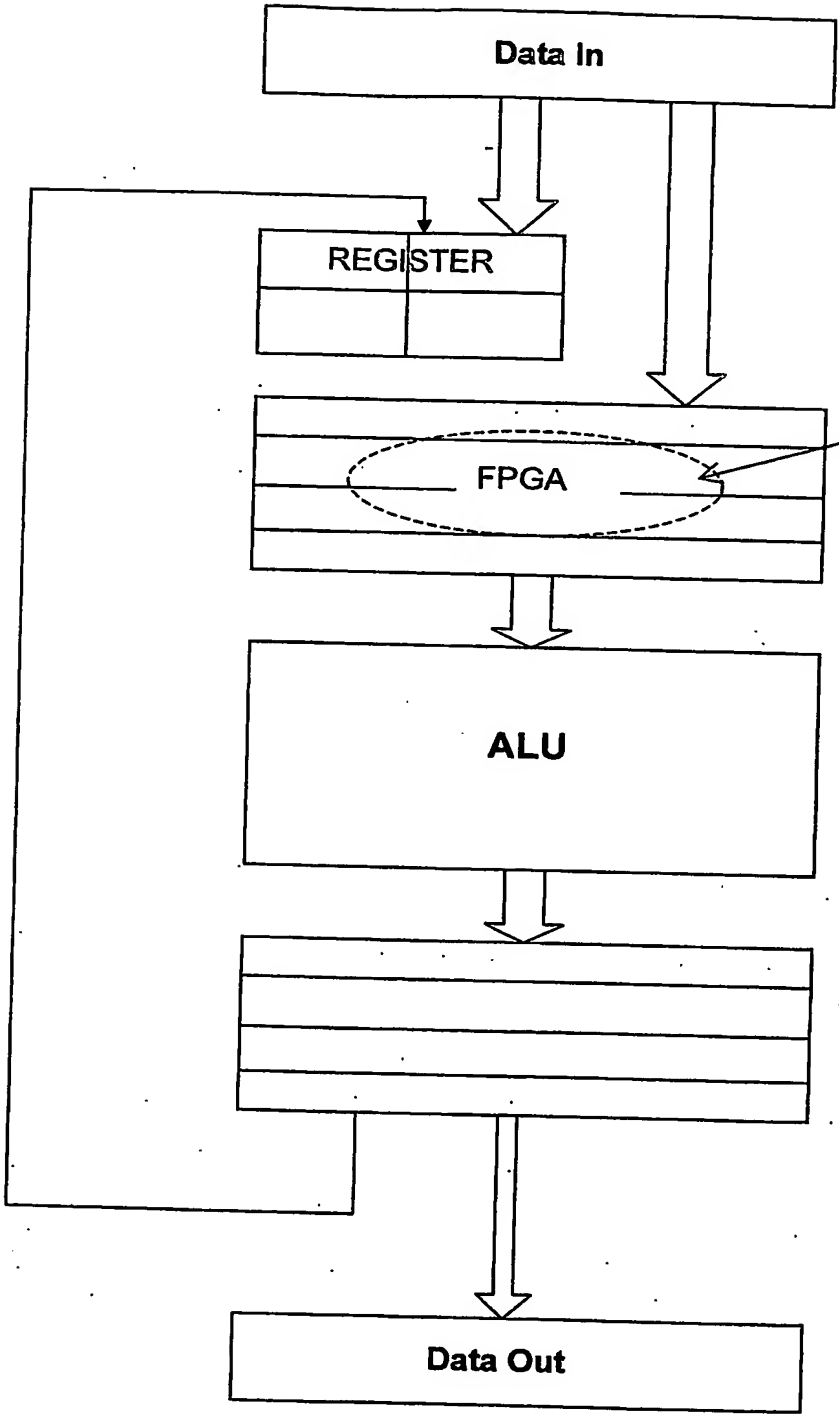


Fig. 3a

Fig. 3 III

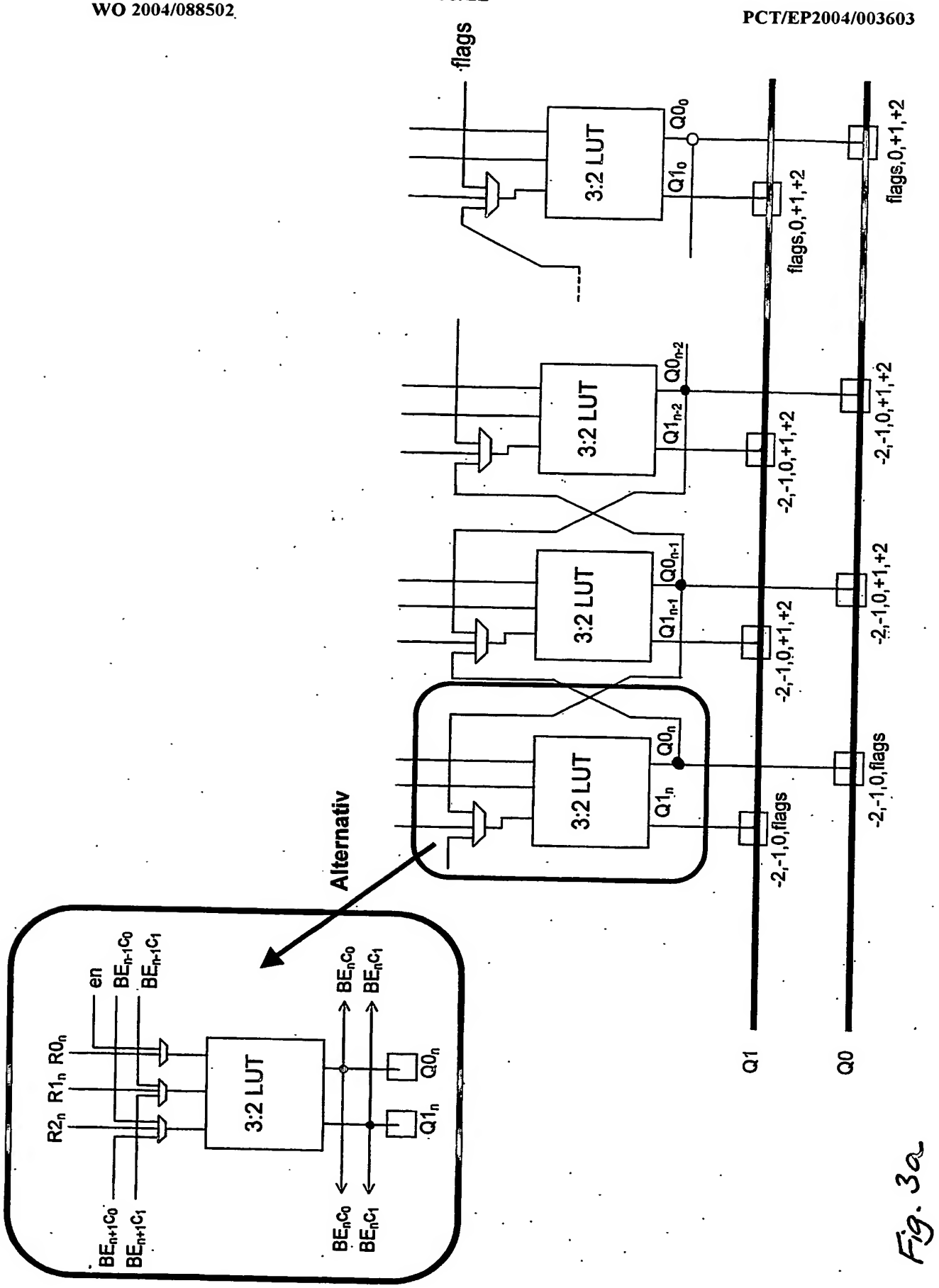
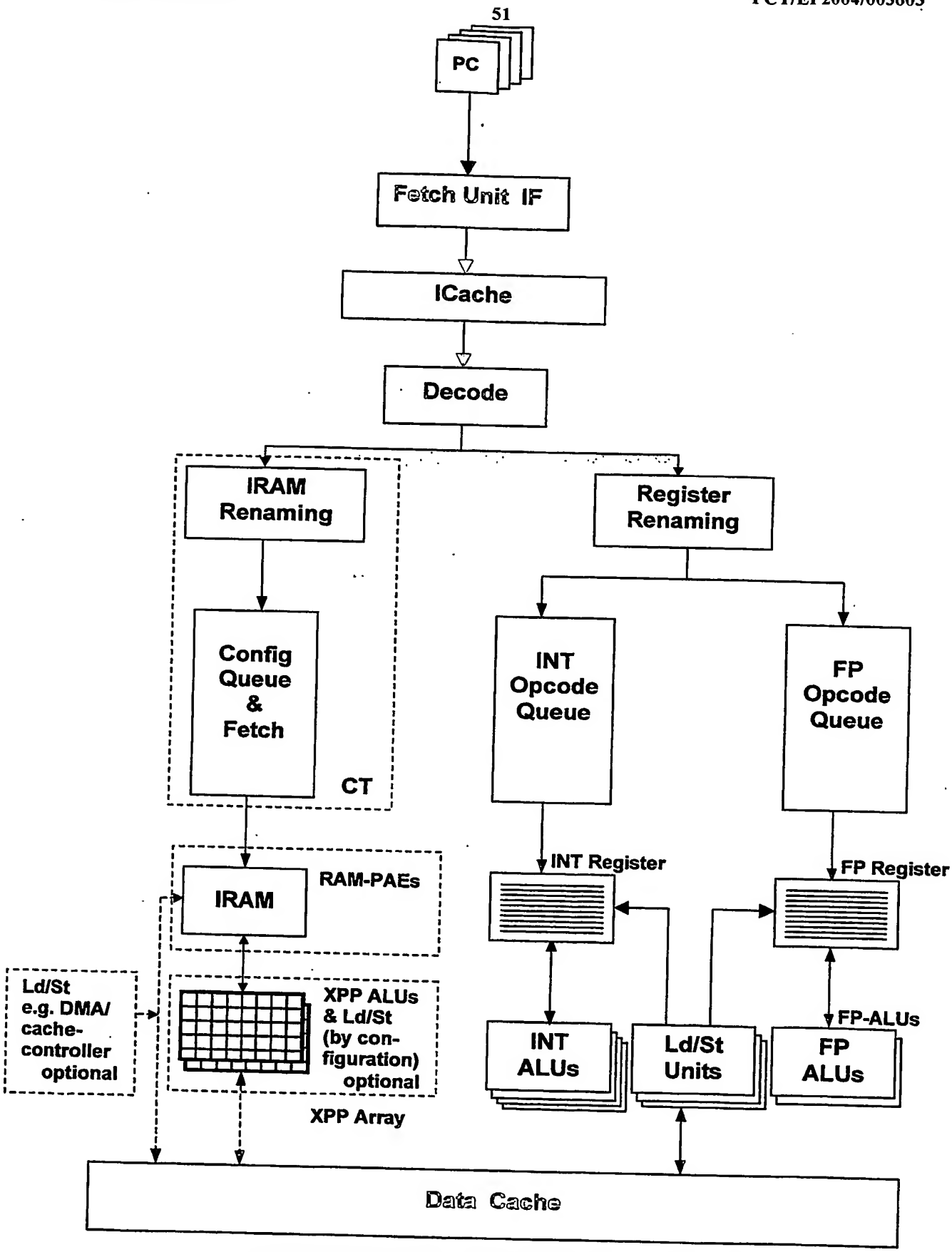

flags
3:2 LUT
Q0₀
Q1₀
flags,0,+1,+2
3:2 LUT
Q0n-2
Q1n-2
-2,-1,0,+1,+2
-2,-1,0,+1,+2
3:2 LUT
Q0n-1
Q1n-1
-2,-1,0,+1,+2
-2,-1,0,+1,+2
3:2 LUT
Q0n
Q1n
-2,-1,0,flags
-2,-1,0,flags
Q1
Q0
Alternativ
en
BEn-1c0
BEn-1c1
R0n
R1n
R2n
3:2 LUT
BEnc0
BEnc1
Q0n
Q1n
BEn+1c0
BEn+1c1
BEnc0
BEnc1

Fig. 3a



Möglicher Aufbau eines SMT-Prozessors mit XPP Thread Resource

Fig. 4a

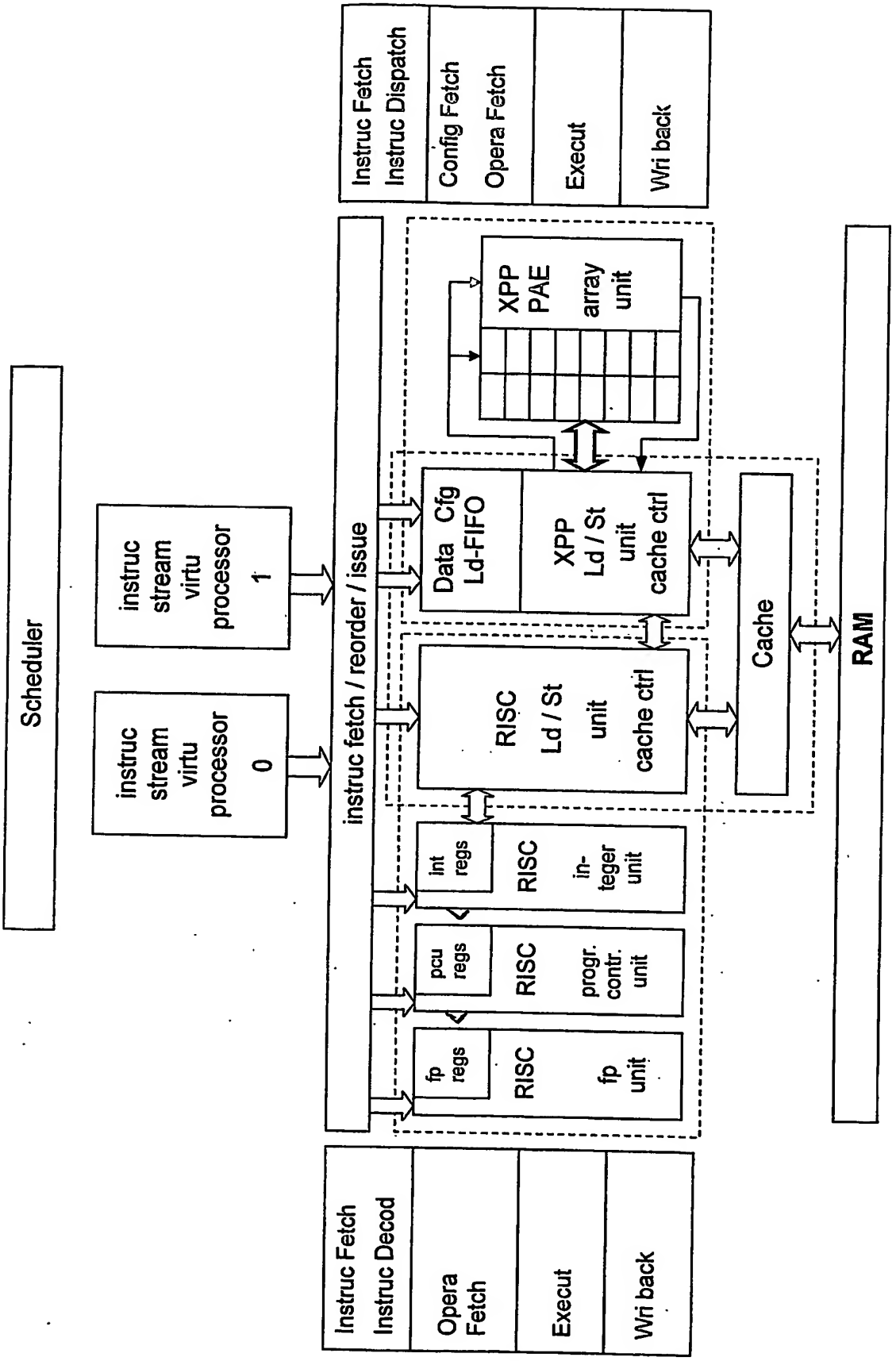

Scheduler
instruc stream virtu processor 0
instruc stream virtu processor 1
instruc fetch / reorder / issue
Instruc Fetch
Instruc Dispatch
Config Fetch
Opera Fetch
Execut
Wri back
XPP PAE array unit
Data Cfg Ld-FIFO
XPP Ld / St unit cache ctrl
RISC Ld / St unit cache ctrl
Cache
RAM
int regs
RISC in-teger unit
pcu regs
RISC progr. contr. unit
fp regs
RISC fp unit
Instruc Fetch
Instruc Decod
Opera Fetch
Execut
Wri back

Fig.

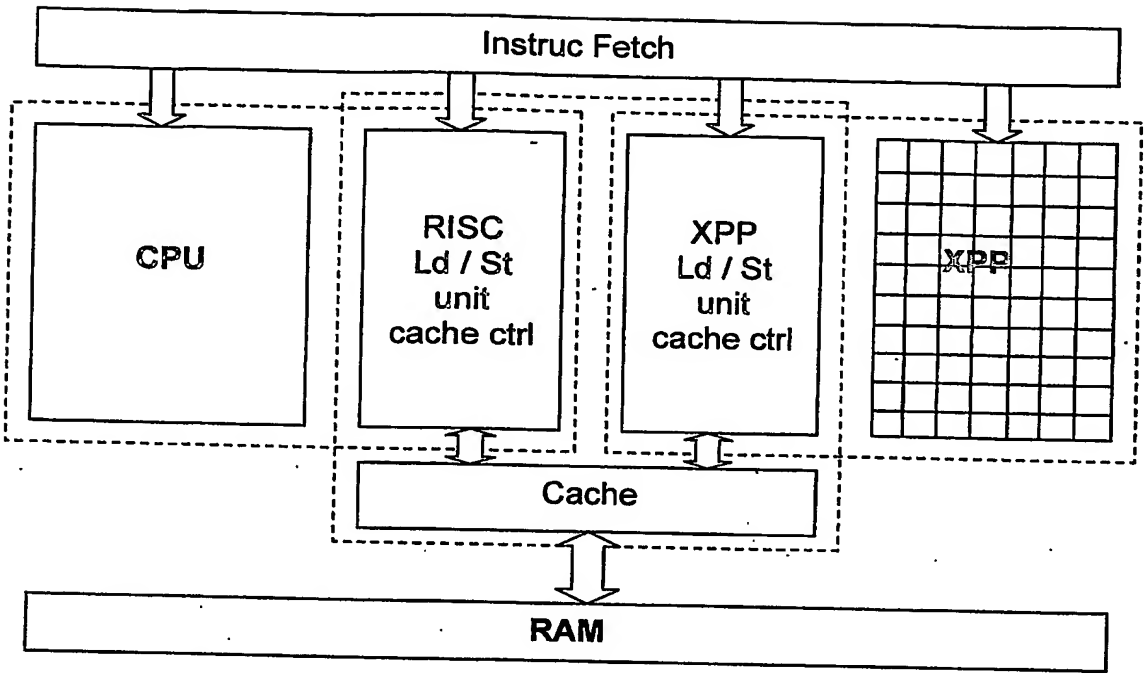

Instruc Fetch
CPU
RISC
Ld / St
unit
cache ctrl
XPP
Ld / St
unit
cache ctrl
XPP
Cache
RAM

Fig. 4C

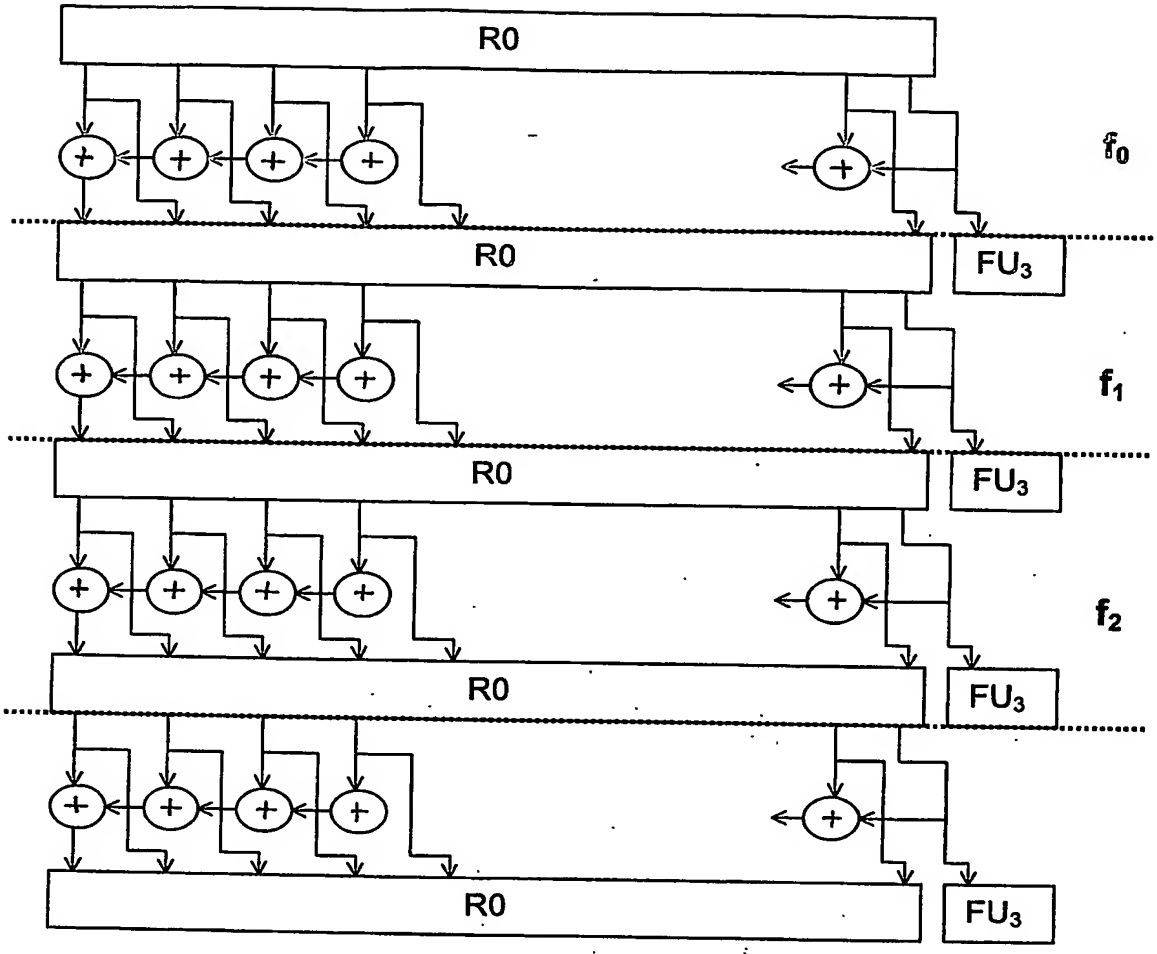


Fig. 5

	Scheduler	MEM-Interface	CT Operation	XPP Operation
1	Task 1	NOP	NOP	NOP
2	NOTask	Ld Task 1 Dat	Config Task 1	NOP
3	NOTask	NOP	NOP	Exec Task 1
4	NOTask	Stor Task 1 Dat	NOP	NOP
5	Task 2	Ld Task 2 Dat	Config Task 2	NOP
6	Task 3	Pre-Ld Task 3 Dat	Preconfig Task 3	Exec Task 2
7	Task 4	Pre-Ld Task 4 Dat Stor Task 2 Dat	Preconfig Task 4	Exec Task 3
8	NOTask	Stor Task 3 Dat	NOP	Exec Task 4

Fig. 6a

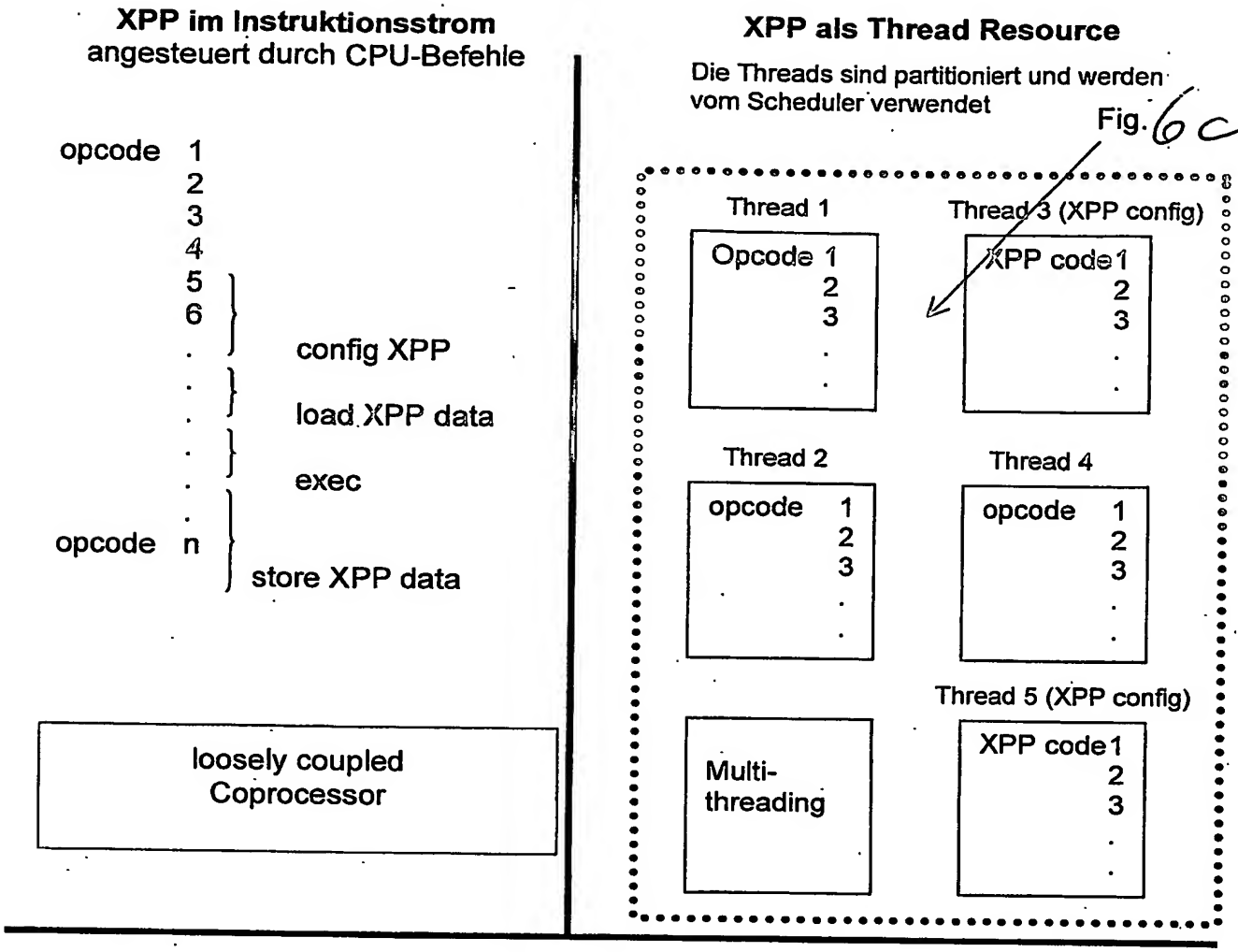

XPP im Instruktionsstrom
angesteuert durch CPU-Befehle
opcode 1
2
3
4
5
6
.
.
.
.
.
opcode n
config XPP
load XPP data
exec
store XPP data
loosely coupled
Coprocessor
XPP als Thread Resource
Die Threads sind partitioniert und werden vom Scheduler verwendet
Thread 1
Opcode 1
2
3
Thread 3 (XPP config)
XPP code 1
2
3
Thread 2
opcode 1
2
3
Thread 4
opcode 1
2
3
Multi-threading
Thread 5 (XPP config)
XPP code 1
2
3

Fig. 6c

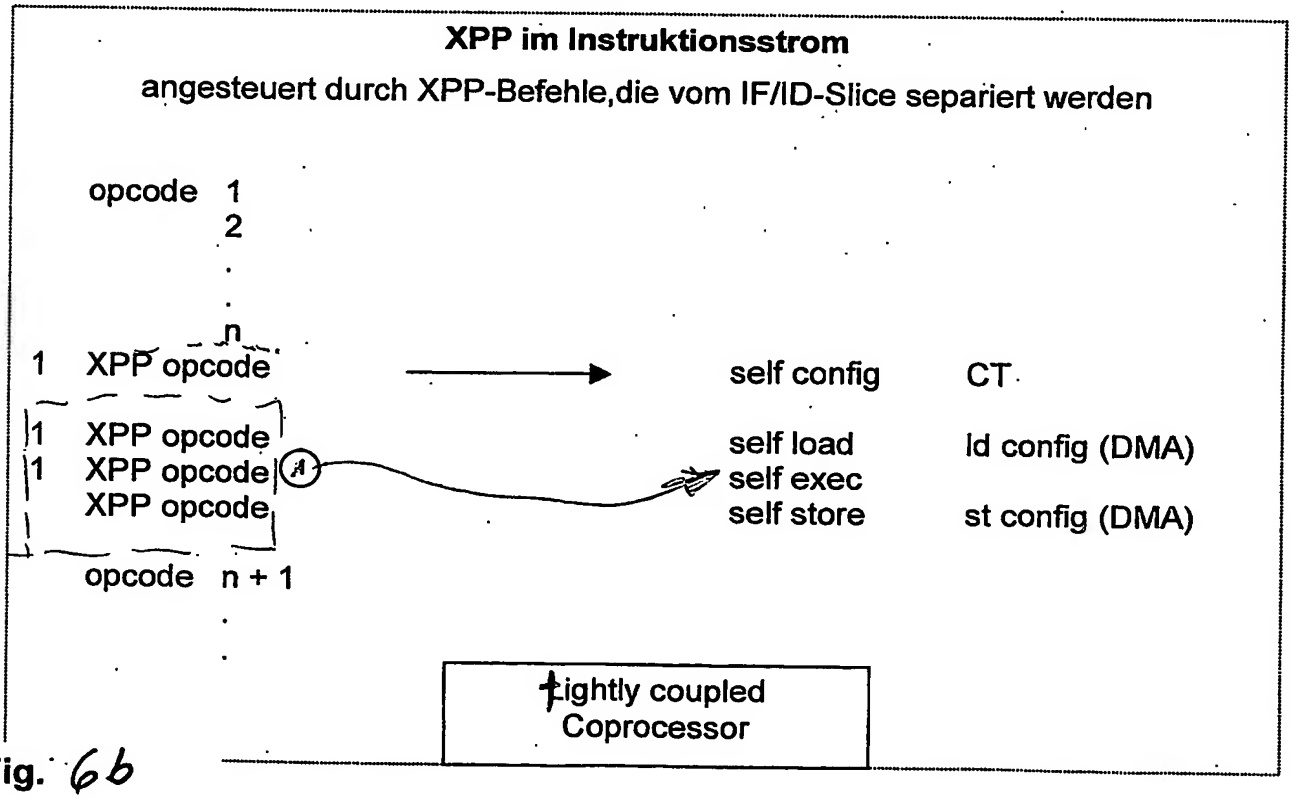
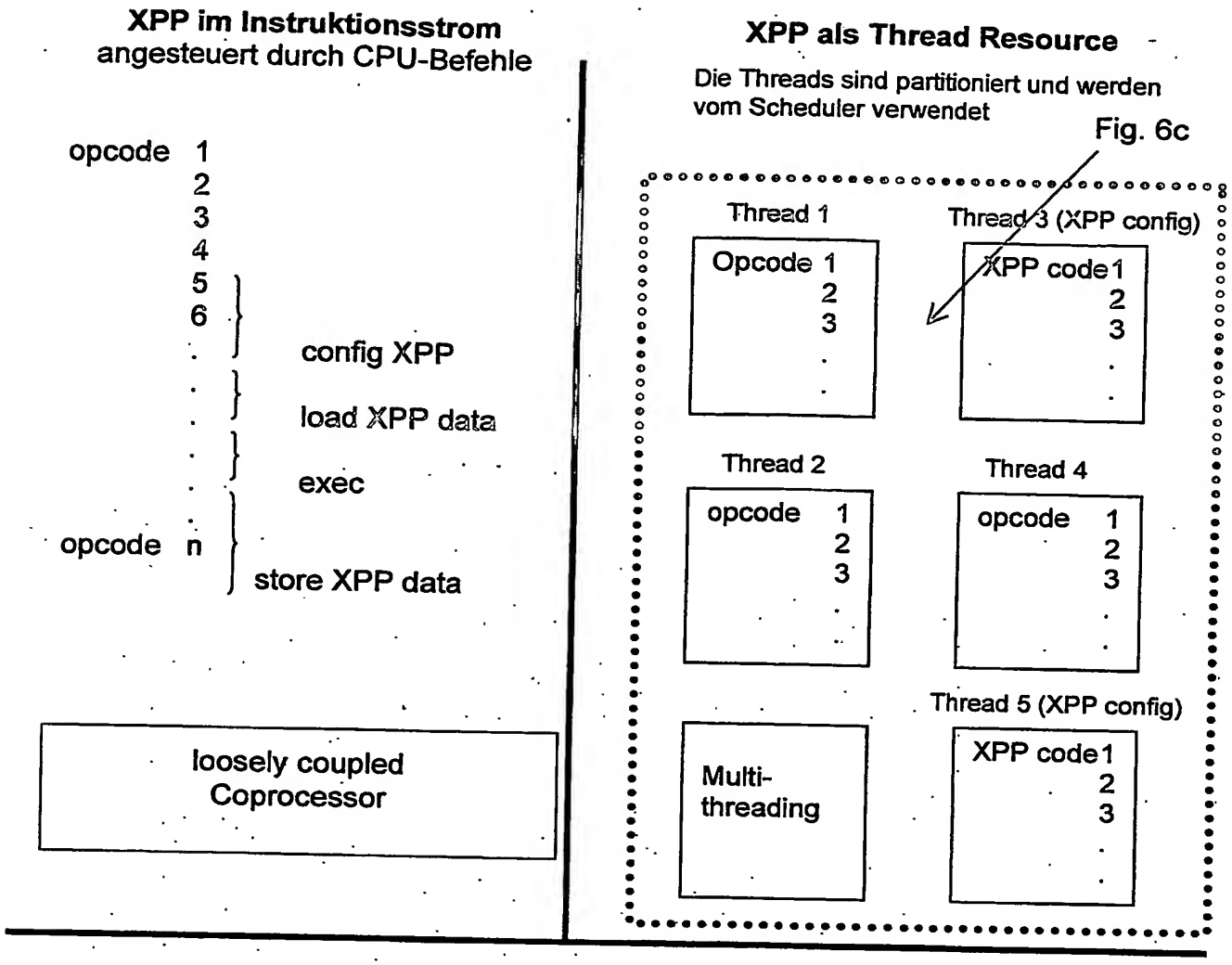

XPP im Instruktionsstrom
angesteuert durch XPP-Befehle,die vom IF/ID-Slice separiert werden
opcode 1
2
n
1 XPP opcode
1 XPP opcode
1 XPP opcode
XPP opcode
opcode n + 1
A
self config
CT
self load
self exec
self store
ld config (DMA)
st config (DMA)
tightly coupled
Coprocessor

Fig. 6b


XPP im Instruktionsstrom
angesteuert durch CPU-Befehle
opcode 1
2
3
4
5
6
config XPP
load XPP data
exec
opcode n
store XPP data
loosely coupled
Coprocessor
XPP als Thread Resource
Die Threads sind partitioniert und werden
vom Scheduler verwendet
Fig. 6c
Thread 1
Opcode 1
2
3
Thread 3 (XPP config)
XPP code1
2
3
Thread 2
opcode 1
2
3
Thread 4
opcode 1
2
3
Thread 5 (XPP config)
XPP code1
2
3
Multi-
threading

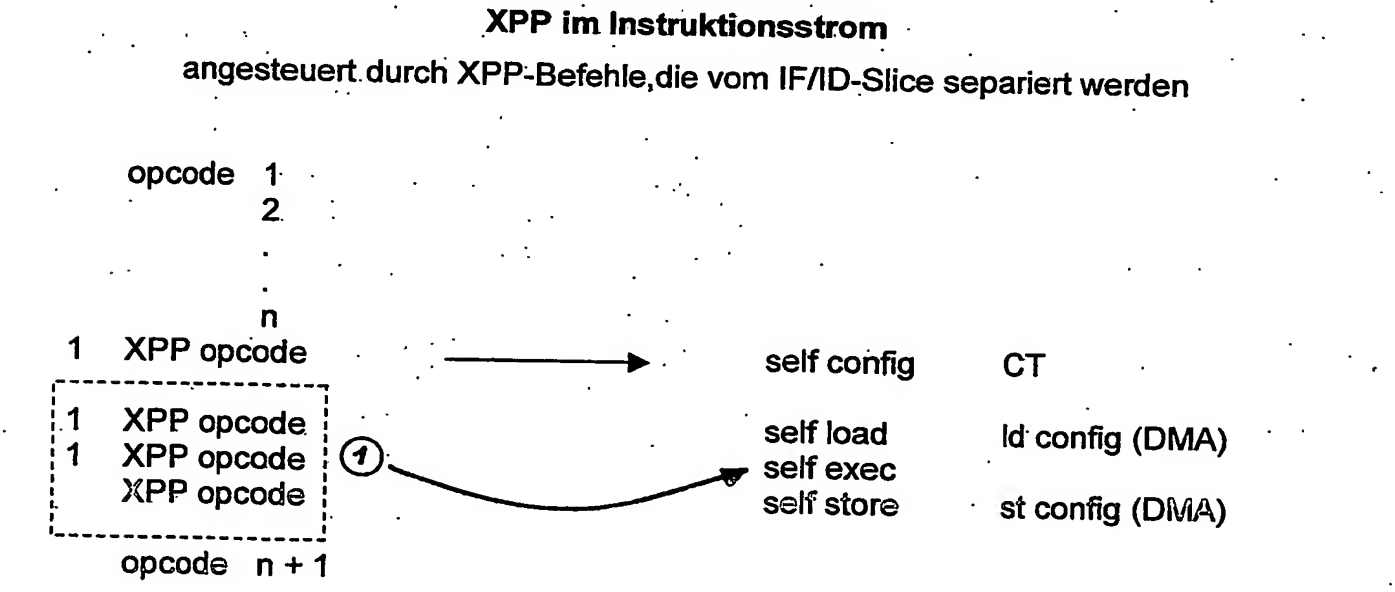

XPP im Instruktionsstrom
angesteuert durch XPP-Befehle,die vom IF/ID-Slice separiert werden
opcode 1
2
n
1 XPP opcode
1 XPP opcode
1 XPP opcode
XPP opcode
opcode n + 1
1
self config
CT
self load
self exec
self store
ld config (DMA)
st config (DMA)
tightly coupled
Coprocessor

Fig. 6b